NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

REAL-TIME APPLICATIONS IN MULTIPROCESSOR SYSTEMS

by

M. Kadri Ozyurt

December 1983

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Real-Time Applications in Multiprocessor Systems	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December, 1983
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) M. Kadri Ozyurt	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943	12. REPORT DATE December, 1983
	13. NUMBER OF PAGES 120
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation	PL/I	Computer graphics
Real-time	RASM-86	
Microprocessor	MDS	
Interrupt	Linked-list	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This thesis builds a simulation model of a tactical fire control system in a real time environment, using a tightly connected multi-processing system consisting of two single board computers. The additional hardware used in this project consists of an ADM-3A video terminal with a built-in retrographics feature, an MDS microprocessor development system, an analog-to-digital converter, and two sets of triplet potentiometers. The potentiometers are used to feed analog information about ownship, targetship, and gun position

to the simulation model, which then evaluates and computes projected target positions and gun control parameters, and displays the results.

Real-Time Applications in Multiprocessor Systems

by

M. Kadri Ozyurt
Lieutenant J.G., Turkish Navy

Submitted in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December, 1983

ABSTRACT

This thesis builds a simulation model of a tactical fire control system in a real time environment, using a tightly connected multi-processing system consisting of two single board computers. The additional hardware used in this project consists of an ADM-3A video terminal with a built-in retrographics feature, an MDS microprocessor development system, an analog-to-digital converter, and two sets of triplet potentiometers. The potentiometers are used to feed analog information about ownship, targetship, and gun position to the simulation model, which then evaluates and computes projected target positions and gun control parameters, and displays the results.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

To this date, many tactical control and decision systems have been designed and implemented in various places, where the nature of the job required fast response and decision making. The NTDS (Navy Tactical Decision System), for instance, is one such system implemented for U.S. Navy ships in 1962, and is still in use today, with recent hardware modifications.

The revolutionary developments in the LSI (Large Scale Integration) and VLSI (Very Large Scale Integration) industries during the period from 1972 to 1983 have made the costs of computing much less expensive than the costs of yesterday's systems. Today's products, which are based on these innovations in large scale integration, have proved to be more reliable and more versatile than the old systems, and they can also be tailored to the needs of whatever the nature of the requirements may be. The serviceability, availability and inexpensiveness of these products, in addition to the above-mentioned features, offer both the designers and the implementers an opportunity to take advantage of this technology.

B. DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than citing each individual occurance of a trademark throughout this thesis, all registered trademarks referred to in this thesis will be listed below, following the name of the firm holding the trademark.

Intel corporation, Santa Clara, California:
Intel, Intel 8086, iSBC 86/12A, MULTIBUS, MDS

Digital Research Corporation, Pacific Grove, California:
CP/M, CP/M-86, PL/I-80, PL/I-86, TED, RASM-86, LINK-86, DDT-86

EX-CELL-O Corporation, Irvine, California:
REMEX Data Warehouse

MicroPro International, San Rafael, California:
Wordstar

Micropolis Corporation, Chatsworth, California:
Micropolis

Lear Siegler, Inc., Anaheim, California:
ADM-3A

C. PURPOSE OF THIS THESIS

The purpose of this thesis is to create a simulation model for real time tactical systems which can be used to study the following features:

1. Multiprocessor system real time performance;
2. System reliability;
3. Graphics Display;
4. Software Engineering.

In order to carry out these objectives, certain hardware changes had to be made. The real time applications necessitate the existence of an interrupt-driven configuration originating from an accurate timer. The interconnections and the appropriate initializations, both in the timing (PIT) and the interrupt (PIC) circuits, were made on the iSBC 86/12A board, to give the required real time clock. After achieving the desired form of operation, the real time executive module (which synchronizes the operation of the simulation system programs) was tested in an interrupt-driven environment. After testing out both the real time executive and most of the simulation system programs, the individual execution times of the simulation system programs were measured with the aid of TIMES.AID, an %INCLUDE file (see Apppendix F).

It was intended to utilize two iSBC 86/12A single board computers in order to study the real time performance of a tightly connected multiprocessing scheme. The Intel MDS (Microprocessor Development System) allows the configuration of such an expansion, through its 20 bit MULTIBUS backplane. It was also planned to write the required software to prevent a single point failure and to gain a "graceful degradation" in the case of a malfunction in any of the single board computers.

The ADM-3A video terminal, with its built-in Retrographics unit, was utilized for graphics display purposes. The Retrographics card contains a Z-80A eight bit microprocessor. This allows the computations for high precision graphics to be done by the video terminal. That improves the efficiency by removing much of the overhead from the iSBC 86/12A single board computers. The Retrographics unit can also make drawings and erasures selectively. This improves the display time, which might be lost due to total erasures and redrawings. In other words, the selective erasing capability decreases the display time, such that the program may make partial erasures, erasures of single objects.

The programs written for the simulation model were made as modular as much as possible, to facilitate testing and maintenance, and to make room for future alterations. A procedure call was placed wherever a critical design decision was to be made. This procedure call gives the option of changing a critical design decision if one that is more efficient is designed. Structured programming and efficient data structures were meant to be utilized. Circular linked-lists are examples of such a programming technique.

D. THESIS ORGANIZATION

The thesis is organized into four chapters. The program listings developed to implement the simulation system are appended at the end of the text. The first chapter covers the background, the disclaimer for the trademarks used in this thesis, the intended purpose of the project and thesis organization. The second chapter covers the system configuration and the hardware components. The third chapter deals with software modules written in both PL/I-86 and RASM-86 assembly languages. The program segments are discussed in detail in this chapter. Some information about the data structures used in the developement of the software are brought up, as well as the initialization of the programmable hardware components. In the final chapter, some conclusions are presented on the work involved in the implementation of the simulation system.

II. SYSTEM HARDWARE

A. SYSTEM CONFIGURATION

The existing system hardware (see Figure 2.1) uses the Intellec microcomputer development system (MDS), which allows the expandability required to set up a multi-processor system. Within the MDS, the boards required for the operation of the system are interconnected through the MULTIBUS backplane. These boards are the following: two iSBC 86/12A boards, the front panel control board, an A to D converter board, and two interface boards for disk drives. The MDS utilizes an Intel disk drive unit which has two disk drives. Standard 8-inch IBM floppy disks are used as the removable storage media.

Each iSBC 86/12A board had a RAM capacity of up to 0FFFF hexadecimal (65535 decimal) eight bit bytes. Since the MULTIBUS is a 20-bit address bus, the address space of the whole system can be expanded up to 1 megabyte. The single board computers can address this memory space by their 20-bit address bus. Each board is so wired that the first 64 Kbyte RAM segment resides on its board (0000 through FFFFH). The 64K RAM segments on each board can be wired to be accessible from the MULTIBUS as dual ported memories in the 1-megabyte address space. One of the iSBC 86/12A boards is the master of the master-slave

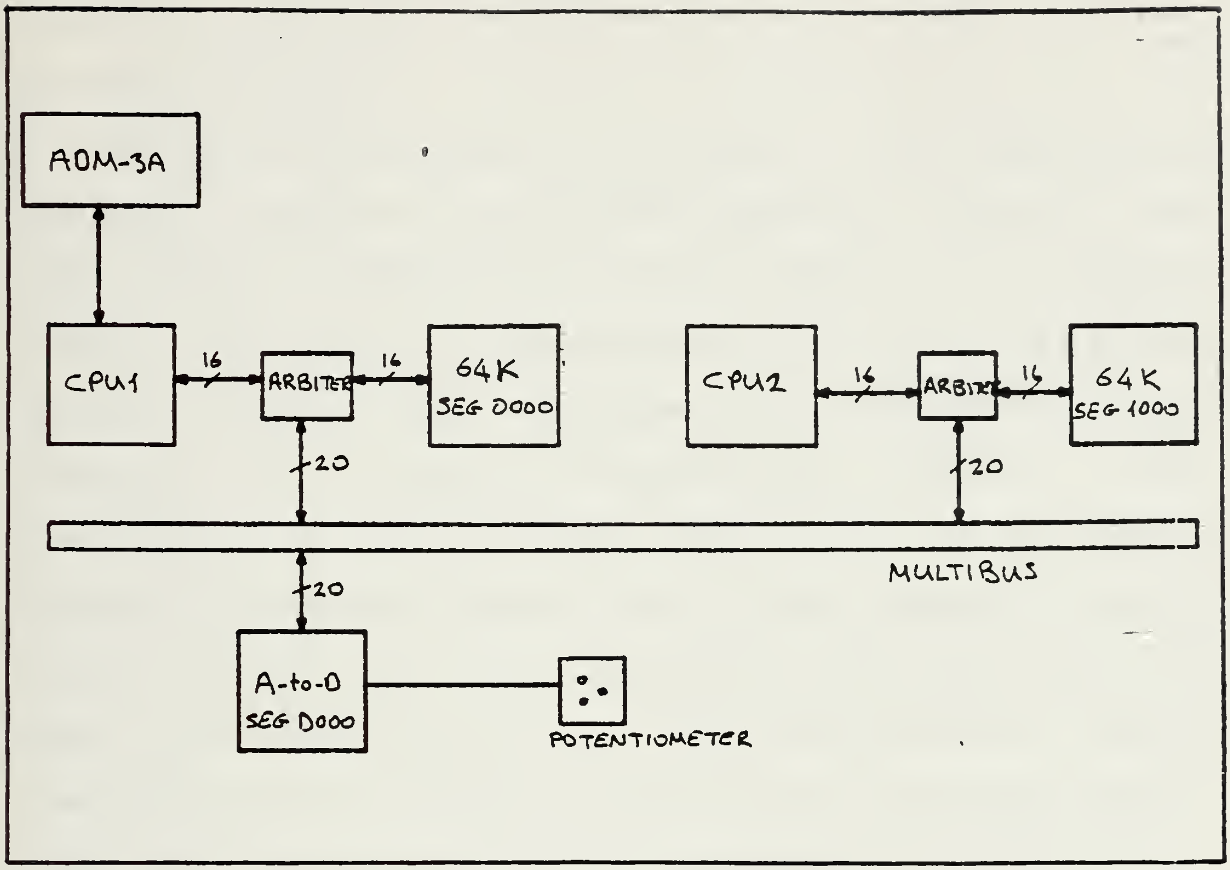

Figure 2.1 - System Interconnection

multiprocessor scheme. The master board is assigned the first 64Ksegment (00000h - 0FFFFh) where the slave board is assigned the second 64K segment (10000h-1FFFFh). In this configuration the A-to-D board occupies the segment with the addresses (D0000h-DFFFFh). The outputs of the A-to-D converter are memory-mapped as eight bit bytes and occupy the locations DF700H through DF70FH. That is to say that they are viewed by the CPU's in the system as being

ordinary memory locations. An analogy to that is that they resemble read only memory that can only be read but not written in to.

The master iSBC 86/12A is connected to the ADM-3A video terminal through it's serial I/O connector. This is the only means of communication of the system to the outside world. The results of the simulation are fed to the video display through this interconnection. The ADM-3A video terminal has a built-in retrographics feature. The consequence of this is that the ADM-3A works not only as an alphabetic terminal but also may act as a graphics device due to the fact that the retrographics card itself has a Z-80A microprocessor built in. This microprocessor allows the high precision graphics computations to be done without the need of any other external processors. In this case, ADM-3A has four operational modes, each one of which has an impact on the simulation graphics and will be discussed later in this chapter.

B. HARDWARE

In the following subsections the individual components that comprise the system hardware are presented.

1. MDS

The Intellec Microcomputer Development System (MDS) is a complete development tool which allows the integration of both microcomputer hardware and software

development. The system operates under the control of an 8086 microprocessor which supervises all system resources such as the main memory, I/O peripheral devices, and optional system facilities, such as A-to-D converters. It can support up to 7 iSBC 86/12A boards in this configuration. Some of the important boards are presented below.

a. Front Panel Control Board

The Front Panel Control Board contains circuits for controlling the front panel options. It also provides some signals for bus control, clock generation, and the bootstrap program. A bus time-out system is included to prevent the CPU from halting operation if a nonexistent memory location or an incorrect I/O port is addressed.

This board produces two types of clock pulses:

(1) Bus Clock (10 MHz), used in Bus transactions;

(2) Common Clock (10 MHz), used by system devices;

b. Disk Interface Boards

These two cards contain the disk controller interface for each drive in the Disk Storage Unit.

2. Single Board Computer

Intel's Single Board Computer iSBC 86/12A is used in the system. It is a member of Intel's complete line of 8- and 16-bit single board computer products and is a

complete computer system on a single printed-circuit assembly. The iSBC 86/12A board includes a 16-bit central processing unit (CPU), 32K bytes (32,768 bytes) of dynamic RAM, a serial communications interface (USART), three programmable parallel I/O ports, programmable timers (PIT), priority interrupt control (PIC), Multibus interface control logic, and bus expansion drivers for interfacing with other Multibus interface-compatible expansion boards. Also included is a dual port control logic to allow the iSBC 86/12A board to act as a slave RAM device to other Multibus interface masters in the system, as is the case in this project. In the current state of the hardware, the RAM capacity of both iSBC 86/12A boards is expanded up to 64K bytes by installing an iSBC 300 Multimodule RAM option. A read only memory of 16K bytes is also added to both iSBC 86/12A boards. The important components that make up the iSBC 86/12A board are discussed in the following subsections.

a. CPU

The iSBC 86/12A Single Board Computer is controlled by an Intel 8086 16-bit Microprocessor (CPU). The 8086 CPU includes four 16-bit general purpose registers that may also be addressed as eight 8-bit registers. In addition, the CPU contains two 16-bit pointer registers and two 16-bit index registers. Four 16-bit segment registers,

specifically: code, data, extra, and stack segment registers; allow extended addressing to a full megabyte of memory. The CPU instruction set supports many variations of addressing modes and data transfer operations, signed and unsigned 8-bit and 16-bit arithmetic including hardware multiply and divide, and logical and string operations. The CPU architecture permits dynamic code relocation, reentrant code, and instruction lookahead.

b. Serial I/O

The serial I/O port is controlled and interfaced by an Intel 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter) chip. The USART is individually programmable for operation in most synchronous serial data transmission formats.

In the synchronous mode, the following are programmable:

(1) Character length

(2) Sync character (or characters)

(3) Parity

In the asynchronous mode the following are programmable:

(1) Character length

(2) Baud rate factor

(3) Stop bits

(4) Parity

In both the synchronous and asynchronous modes, the serial I/O port features half- or full-duplex, double buffered transmit and receive capability. The USART transmit and receive clock rates are supplied by a programmable baud rate/time generator.

c. Programmable Interval Timer

Three independent, fully programmable 16-bit interval timer/event counters are provided by in Intel 8253 Programmable Interval Timer (PIT). Each counter is capable of operating in either BCD (binary coded decimal) or binary modes; two of these counters are available to the system's programmer to generate time intervals under software control. In this thesis project the counter1 is used to generate timing pulses required for the real time clock operation to the system software. These pulses are sent to the PIC via the interrupt matrix as being an IR1 input request.

d. Priority Interrupt Control

The priority interrupt control (PIC) which can be programmed to respond to edge-sensitive or level-sensitive inputs, treats each true input signal condition as an interrupt request. After resolving the interrupt priority, the PIC issues a single interrupt request to the CPU. Interrupt priorities are independently programmable under

under software control. The programmable interrupt priority modes are:

(1) Nested Priority. Each interrupt request has a fixed priority: input Ø is highest, input 7 is lowest. This mode of operation is chosen in this thesis project;

(2) Fully Nested Priority. This is essentially the same as item (1) above, with the exception that the requesting input is not locked out and pending requests are still accepted;

(3) Auto-Rotating Priority. Priorities are equal. The last received input becomes the lowest priority input;

(4) Specific Priority. Software assigns the priorities;

(5) Special Mask. Interrupt requests that are being serviced are masked out;

(6) Poll. The CPU's internal interrupt enable is disabled. Interrupt service is achieved by a programmer-initiated Poll command.

The iSBC 86/12A board provides two sorts of interrupts which are bus vectored (BV) and non-bus vectored (NBV). The former deals with the interrupt requests from off-board sources where the latter deals with various on-board sources. The interrupt requests are fed to the

under software control. The programmable interrupt priority modes are:

(1) Nested Priority. Each interrupt request has a fixed priority: input Ø is highest, input 7 is lowest. This mode of operation is chosen in this thesis project;

(2) Fully Nested Priority. This is essentially the same as item (1) above, with the exception that the requesting input is not locked out and pending requests are still accepted;

(3) Auto-Rotating Priority. Priorities are equal. The last received input becomes the lowest priority input;

(4) Specific Priority. Software assigns the priorities;

(5) Special Mask. Interrupt requests that are being serviced are masked out;

(6) Poll. The CPU's internal interrupt enable is disabled. Interrupt service is achieved by a programmer-initiated Poll command.

The iSBC 86/12A board provides two sorts of interrupts which are bus vectored (BV) and non-bus vectored (NBV). The former deals with the interrupt requests from off-board sources where the latter deals with various on-board sources. The interrupt requests are fed to the

PIC through the jumpers of the interrupt matrix, which will be discussed in the next subsection.

e. Interrupt Matrix

Interrupt requests may originate from eighteen sources without the necessity of external hardware. The interrupt matrix connects the selected source lines to a maximum of eight selected inputs of the PIC. It is an array of pins which can be connected to each other via jumper wires. There are two types of pins. The eighteen source lines constitute the input pins, where the pins that lead to the IR0 through the IR7 inputs of the PIC constitute the output pins. (See Figure 2-2 for the interconnection scheme of this thesis project.)

f. Dual Port RAM

The iSBC 86/12A board has an internal bus for all on-board memory and I/O operations. Hence, local (on-board) operations do not involve the MULTIBUS interface, making it available for other iSBC 86/12A boards for a multi-processor scheme. Dual port control logic is included to interface so that the iSBC 86/12A board can function as a slave RAM device (or common memory) when not in control of the Multibus interface. The CPU has priority when accessing on-board RAM. After the CPU completes its read or write operation, the controlling bus master is is allowed to access RAM and complete its operation. Where both the CPU and the controlling bus master have the need

to write or read several bytes or words to or from the on-board RAM, their operations are interleaved. For CPU access, the on-board RAM addresses are assigned from the bottom up of the 1-megabyte address space; i.e., 00000-0EFFFh. The slave RAM address decoding logic includes jumpers and switches to allow positioning the on-board RAM into any 64-K segment of the 1-megabyte system address space. The slave RAM can be configured to allow either 16K, 32K, 48K, or 64K access by another bus master, with the installation of the iSBC 300 Multimodule RAM. In this thesis project all of the 64K-byte memory of the slave iSBC 86/12A is made accessible to the master. Furthermore, both iSBC 86/12A boards are configured to occupy the first 128K section of the 1-megabyte address space.

3. <u>A-to-D Converter Board</u>

This board is electrically and mechanically compatible with any iSBC 86/12A board and with MDS. Both the anolog input and output systems are contained on a single printed circuit board that is treated as ordinary memory locations by the CPU (memory mapping). This board simply gets the analog signals form the potentiometers and converts them to the digital signals compatible with TTL standards. The output of the A-to-D converter is one byte per potentiometer input, which varies from +127 to -128. This is the maximum value range a fixed binary (7) variable can assume in the PL/I language, by definition. So, the

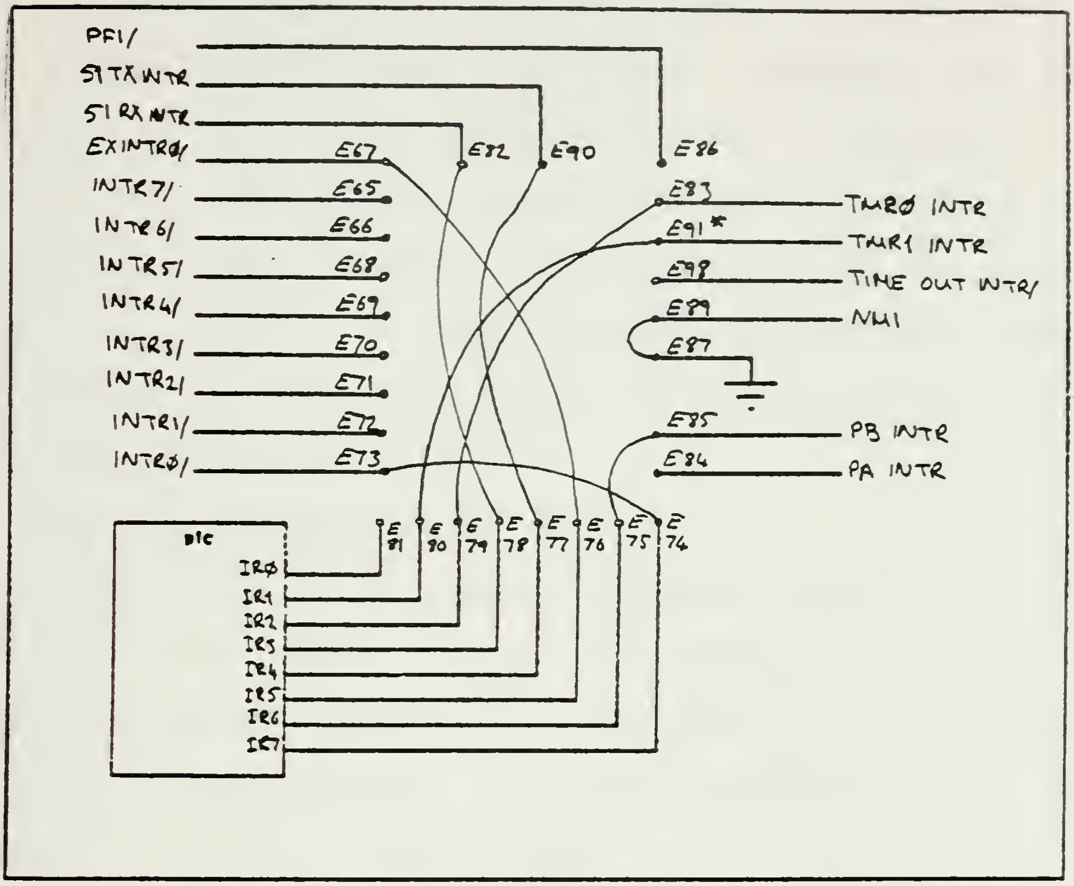

Figure 2.2 -Interrupt Matrix Interconnection

software system thinks of the potentiometer value as changing between +127 and -128. The A-to-D converter board is configured to occupy the segment D000: (14th 64K portion of the 1-megabyte address space).

4. ADM-3A Terminal

This terminal is the only means by which the operator communicates with the system. It is connected to the system with the master iSBC 86/12A board's serial I/O connector. It is an interactive device which is used to enter, display, and send information to a host computer,

and to receive and display information from that computer. The information exchange between the terminal and the computer is made at different baud rates, ranging up to 19200. In this scheme, a 9600 baud rate is used. The keyboard contains 59 keys. The display memory is a RAM which is capable of holding 1920 characters. Data characters are displayed on 24 or 12 equally-spaced rows, each consisting of 80 columns.

5. RG-512 Retrographics Card

The RG-512 Retrographics printed circuit board is added to the ADM-3A terminal to extend the data and graphics display capabilities with the aid of a Z-80A built-in microprocessor. The RG-512 employs the bit map method of storing graphic images. This information is stored in a digital memory as a rectangular array of bits. Each bit is mapped onto the CRT screen and can cause a bright spot to be displayed. The RG-512 displays graphs and pictures by writing the proper bits into the graphics memory. One of the important features of the RG-512 is the ability to erase portions of the screen selectively. This is desirable when the application requires the use of dynamic displays employing motion or rotation to convey information. The RG-512 has four modes of operation. These are the ADM-3A Alpha Mode, the 4010 Alpha Mode, the Point Mode, and the Vector Mode. The first one is equivalent to the operation of ADM-3A without RG-

512. The latter three modes make use of the bit map method.

6. Potentiometers

Two sets of triplet potentiometers are used as simulating analog sensor information sources. They feed anolog signals to the input of the A-to-D converter board, varying between -5V and +5V.

III. IMPLEMENTATION OF THE SOFTWARE

A. GENERAL INFORMATION

1. Modularity

A modular and extensible simulation program is aimed at simplifying the debugging and testing phase and at facilitating possible alterations. The hierarchy of the modularity is composed of a head module and four second-level modules connected to the head module. These second-level modules are the initialization module, the simulation system module, the real time executive module, and the dynamic debugging tool module. These modules are separately compiled PL/I-86 and RASM-86 programs. Each main module is further subdivided into third-level modules to gain a finer granularity of modularity. In the programs, two useful special features of PL/I-86 are used. Those are %INCLUDE and %REPLACE statements. By those statements, global declarations that are the same in the scope of the simulation program need not be declared within each and every module. Instead, they are grouped together in the GLOBALS.INP declaration file. The %REPLACE statement allows constants to be declared as in the other high-level languages like Pascal, such that the value of the constants can be changed without having to go through every program segment in which they occur.

2. Data Structures

Linear arrays and arrays of structures (records) are used in the simulation program as data structures. These data structures are then linked to each other to establish circular linked lists. Figure 3.1 (on the next page) explains the general picture of the circular linked lists in the simulation program. Fixed size data structures with fixed binary pointers are used in the program, rather than pointer data, to avoid the dynamic system overhead and to retain the benefit of random access capabilities inherent to linear arrays.

The structure SHIP has two pointers, in addition to the fields that hold specifications about the ships in the area. Those pointers, PTR and LINK-SHIP have different purposes for different ships. SHIP (1), for instance, being the ownship points to two different circular linked lists. PTR points to the enemy ship's circular linked list by pointing to the target ship which is engaged (tracked) for the sea battle. The other pointer, LINK-SEIP points to the friendly ships circular list. The PTR field of other ships, on the other hand, points to another circular linked list, WAKE, to record their past positions that will be used for tracking and display purposes, where LINK-SIP points to the other ships in their category. The reason for using circular lists in this program is the ease with

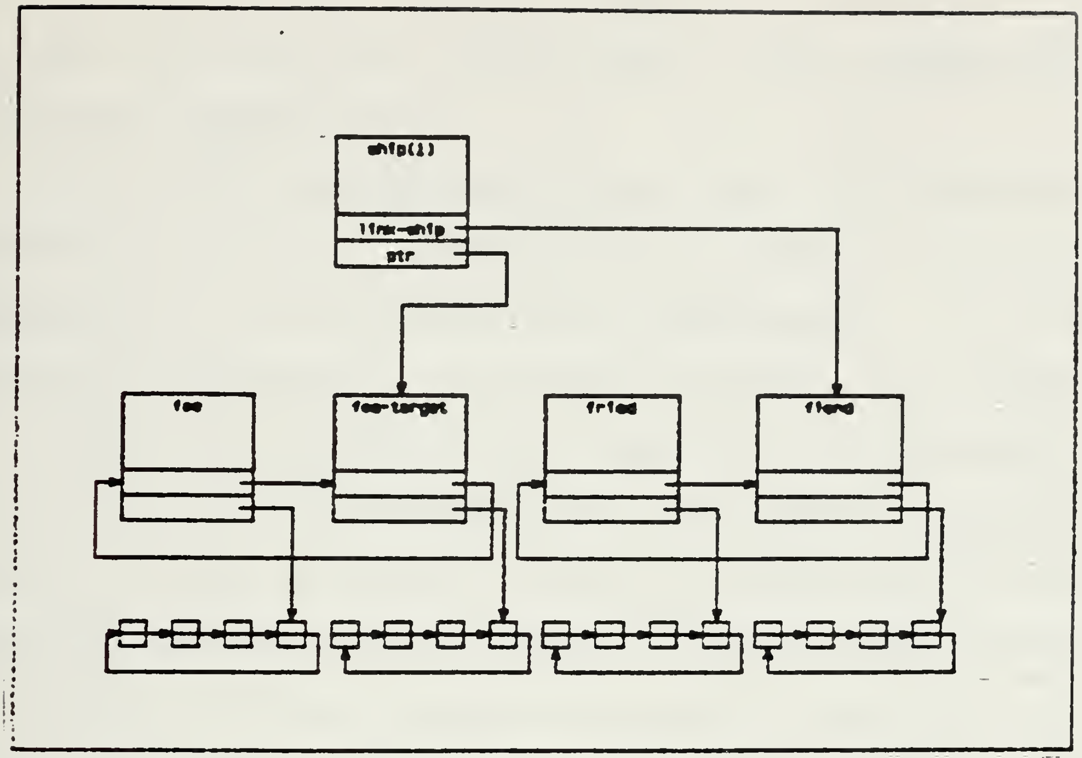

Figure 3.1 The structure of the circular linked lists

which one traverses through the lists and which does not necessitate the use of another external pointer.

3. Other Features

The PL/I exception handlers (ON body statements) are used extensively in the dynamic debugging module, and in various interactive parts of the program, to intercept the error conditions that might be raised during the testing and execution of the program. The ON body statements are contained in the ERRHAND.AID file, which is an %INCLUDE file. Upon receiving the control through a raised error condition, the statements in the file prompt

the user and give the control over to the REENTRY.PLI interactive debugging tool.

One of the exceptional features of the simulation program is the use of non-local goto statements which are unacceptable in structured style of programming. It is an inevitable requirement, by the PL/I language, to suppress the raised error conditions by a non-local goto statement. It is also used in some parts of the dynamic debugging module, in order to by-pass the flow of control over to the debugging program when the optional ERRORON boolean switch is closed. Explicit comments are offered wherever non-local goto statements are used, to avoid confusing the reader.

B. SOFTWARE FUNCTIONAL DESCRIPTION

In the following sections, the structure of the modules and the programs that belong to those modules are described. The program listings are presented as appendices.

1. Head Module

a. WAR.PLI

This main procedure is the head node of the hierarchical structure of the procedures used to modularize and structure the implementation of the simulation program. It contains two call statements, one of which is to the initialization module, to set up the tactical database

and to initialize various external variables that are used throughout the simulation program. The other call is for passing the control to the real time executive module where the control stays for the rest of the program execution. A listing of WAR.PLI is presented as Appendix A.

2. Initialization Module

a. INITVARS.PLI

This PL/I routine, when called from the main procedure, constructs the tactical database in an interactive manner. It first initializes the pool of available SHIP and WAKE modes for later use. It then gets the interval of time which is used to update the information about ships in the tactical area and other time dependent functions. This time interval must be equal to the period of the real-time interrupts which depend on the timing constants used during the initialization of PIT. Detailed information will be presented later under the hardware initialization section. Then the control proceeds to establish the tactical database interactively. In this session, the initial information about azimuth, range, friend or enemy are written to the proper fields. Then, a circular linked list of four nodes is composed and the PTR field is made to point to that list. Finally, dependent on whether friend or foe, that particular node is added to the appropriate circular linked list. INITVARS.PLI makes use of various internal subroutines for linked list

operations. Those subroutines simply extract a node from the pool of available nodes. After establishing the linked lists, the INITVARS.PLI initializes external variables that are used throughout the program. A listing of INITVARS.PLI is presented in Appendix B.

3. Simulation System Module

This module is composed of four PL/I programs which perform the simulation under the control of the real-time executive module. The following subsections describe the functional description of those programs. The listings of the program segments that comprise this module are presented in Appendix C.

a. TACTICAL.PLI

This routine has the highest priority among the system module programs. It first updates the position of each ship in the tactical area by calculating the relative velocity and multiplying that with the time interval, which is the period of timing interrupts that occur every 250 milliseconds. The control then proceeds to calculate the future positions of the ships, for those ships which have been in the area for more than 4 seconds, and which are included in the enemy ship circular linked list. The routine uses the polynomial least squares curve fitting method with Legendre Polynomials. The coefficients are pre-calulated for the position of the ship one second after the time of calculations, based on the past four wake

points. TACTICAL.PLI also calculates the trajectory of the travelling projectile if the gun is fired.

b. DISPLAY.PLI

This routine, which is invoked every second, is the interface of the simulation program for the ADM-3A screen. It simply traverses the circular linked lists and generates the appropriate display objects for the ships, their aim points which the own ship aims at, and the gun aim point. It also displays the travelling projectile if it is fired by the system. The routines for generating the objects are internal for the DISPLAY.PLI. Another routine, TRANSLATE, translates the cartesian coordinates to the stream of characters that represent the x and y grid coordinates of the RG-512 Retrographic Screen Memory. Finally, the internal routine DRAW puts the generated objects on the screen in vector mode. According to the key variable, D, it either sets the data level to white and puts the object on the screen, or sets the data level to black and makes selective erasures.

c. STATUS.PLI

This routine, which is invoked at every second, is the interface of the system to the ADM-3A video terminal keyboard. The commands for the system are read from the keyboard by calling the serial I/O chip (USART) interface assembly program, KEYBOARD, which will be presented under Miscellaneous Assembly Routines.

The are four boolean variables used in the simulation program. Those are ENGAGED, MAGNIFIED, FIRED, and ERRORON. The STATUS.PLI sets these variables according to the value of the parameter passed to the assembly interface routine. ENGAGED (with "E") shows if the system is engaged to any target (for TACTICAL.PLI). MAGNIFIED (with "M"), which is used in DISPLAY.PLI, shows which display scale is being used and what the reference point of the display is. Usually, the display on the screen is relative to own ship. But that can be changed so that the ship engaged is at the center to the screen by setting MAGNIFIED true. FIRED (with "F") boolean variable is used to commence the ballistic calculations and display. It is used both in TACTICAL.PLI and DISPLAY.PLI. Finally, ERRORON (with "D") is used to transfer the control to the dynamic debugging module.

d. IDLE.PLI

This program is the idle routine for the system, as it waits for a 250 millisecond timing interrupt to occur. It is the interface program for the six potentiometers which are used as sensors for the own ship velocity vector, the known ship velocity vector, and the gun elevation and bearing. The velocity vectors are composed of speed and course components. The IDLE.PLI gets this information by calling the Analog-to-Digital Converter interface assembly program, ATOD. The control then

proceeds to convert this information, which is in the fixed binary (7) form in range (-128, +127), to appropriate coordinate values, e.g. 50 knots maximum speed and the true azimuth between 0 and 360 degrees. This routine calculates own ship velocity components in cartesian coordinates for later use by TACTICAL.PLI. It also makes the initial ballistic computations for the gun.

4. Real Time Executive Model

This module works as the interrupt handler for the real time interrupts that are initiated by programmable hardware components every 250 milliseconds. It is invoked by the WAR.PLI the first time and interrupts thereafter. It responds to the timing interrupts, which tell the system that data must be collected at this point in time. The module then resolves the priorities of the simulation system module programs and arbitrates the flow of control during the execution of the system. The real time executive module makes use of the operating system primitives, which are presented in the following subsections. The labels P1 through P4 are associated with the simulation system modules TACTICAL.PLI, DISPLAY.PLI, STATUS.PLI, and IDLE.PLI, respectively. The listing of the programs included in the Real Time Module are presented in Appendix D.

a. ARBITER.A86

This assembly language program is the real "workhorse" of the entire system. It first allocates stack areas for four simulation system during the assembly time. Upon invocation by the main procedure, it initializes the programmable hardware components and transfers the control to the P4.PLI process, which in turn calls IDLE.PLI repetitively until the first timing interrupt occurs. The interrupt entry point PROC0, where the process switching starts, is entered by the interrupt software. At this point of execution, the external variable fourthevc, which signals the system that 250 millisecond event has occured, is updated to the new value by incrementing it by one. After storing the state of the program which is interrupted during its execution, ARBITER.A86 invokes SCHEDULE.PLI to obtain the name of the ready program that has the highest priority. If there are none, the interrupted program is resumed. If there is any ready process of higher priority, then ARBITER.A86 loads the process state and gives the control over to it. During this process switching, the upper boundaries of the stack areas are checked for a possible stack overflow, which could happen if the time interval was not sufficiently large for the system module routines to finish execution before the next timing interrupt comes. ARBITER.A86 has a second entry point, STORESTATUS, for the synchronization primitive AWAIT.PLI to

enter when the correct number of interrupts for the calling synchronization primitive have not yet occured.

b. AWAIT.PLI

This synchronization primitive is invoked as as an operating system primitive, by any process, P1 through P3. AWAIT compares the value of the external variable FOURTHEVC to the threshold value of the calling process to see if it is greater or equal to the value at which the process is to proceed. If not, then it calls the STORESTATUS entry of ARBITER.A86 to relinquish the control to the awaiting ready process, or to the P4.PLI that calls the IDLE.PLI simulation system program as the system idling routine.

c. SCHEDULE.PLI

This synchronization primitive is called by ARBITER.A86 to return the name of the highest priority ready process. It does that simply by identifying the first ready process on the list. Because the scheduler scans the list in the descending priority order, the highest priority ready process will automatically be scheduled.

d. THRESH.PLI

This routine, when invoked by processes P1 through P3, increments the corresponding thresholds in an external one dimensional array called THRESHOLD. This table is used by the AWAIT.PLI and SCHEDULE.PLI

synchronization primitives to decide whether or not a process is ready for execution.

e. P1.PLI

This process is basically an infinite loop. Within this loop, there are three subroutine calls. The routine first makes a call to AWAIT.PLI to see if it is time for it to proceed. If not, the control doesn't come back again; instead, the current state of the process is stored by ARBITER.A86 and the highest priority ready process is executed. If it is the time, the control proceeds to call the simulation system module program TACTICAL.PLI. After that, a call to THRESH.PLI is made, where the threshold value that is allocated to P1.PLI is incremented by the proper value. When the infinite do loop repeats itself, the call to the AWAIT.PLI will indicate that the process TACTICAL.PLI is not yet ready for execution and control is transfered to the highest priority ready process.

f. P2.PLI

This process is identical to P1.PLI in form except the call is to DISPLAY.PLI instead of to TACTICAL.PLI.

g. P3.PLI

This process is identical to P1.PLI in code except the call is to STATUS.PLI.

h. P4.PLI

This process is similar to P1 through P3 in the structure described above. There is only one call in the infinite do loop, which is to IDLE.PLI. This routine is always ready for execution and, basically, repeats itself until the next timing interrupt comes along.

5. Miscellaneous Assembly Routines

There are few machine dependent functions that cannot be accomplished by the high-level language PL/I-86. Assembly routines were written to interface the PLI-86 programs with the hardware of the 8086 microprocessor. These assembly routines are included within the main body of ARBITER.A86. There are two parameter passing conventions from PL/I-86 to the assembly language routines. In the first one, there is only one argument passed in the accumulator, as in a function call. In the subroutine calls, which is the case here, the address of the VECTOR that contains the pointers to the actual parameters is passed in the BX register. The following subsections give some descriptions about those assembly routines. Appendix E shows the listings of the modules.

a. KEYBOARD.A86

This routine is invoked by STATUS.PLI to read the keyboard. It is written so that the keyboard status is read to see if a key had been pressed instead of waiting indefinitely until a key was depressed, as would be the

case had the PL/I get statement been used. The program first reads in the status of the serial I/O interface chip (USART) to see if a character has been received from the keyboard. If it has, then the character is read and placed into a corresponding character variable, which is the formal parameter in the subroutive invocation. If there is no character received since the last attempt to read, the ASCII equivalent of Ø is put in key. The reason for that is that the character Ø is not being used as a command.

b. ATOD.A86

This assembly routine is called by IDLE.PLI to read the first six Analog-to-Digital Converter Board outputs. The reason to write this assembly routine is that the Analog-to-Digital Converter ports are memory mapped to be in the segment DØØØH. The PL/I function UNSPEC works for those memory locations which are included within the first 64K bytes of memory. The assembly routine sets the proper segment and source index registers to point to those locations and makes an ordinary read operation. This value is then put in the formal parameter passed to the PLI-86 routine.

c. RINGBELL.A86

This assembly routine simply sends a bell character to the video terminal and causes a bell sound to ring. This is equivalent to sending a control G in PL/I-86.

The only difference is that it can be used in other assembly routines.

d. WAIT.A86

This assembly routine reads in the status of the I/O interface chip and waits until the transmitter buffer is empty; i.e. the character which had been in the buffer is received by the video terminal and an acknowledgement signal is sent back to the interface chip. This routine is used by assembly routines that put out a message.

e. SUSPEND.A86

This routine simply resets the interrupt bit of the program status word (PSW) to disable the 8086 CPU from acknowledging the interrupts. It is used by the dynamic debugging system to stop the real time clock.

f. RESUME.A86

This routine first sets the interrupt flag to enable the 8086 CPU to respond to the interrupt requests. It also resets the PIT that is used to generate the timing clock pulses for the PIC.

D. INITIALIZATION OF THE PROGRAMMABLE COMPONENTS

The iSBC 86/12A board has three programmable hardware components, which were described in Chapter II. In the following subsections, the initialization sequences for

those hardware components which produce the real time synchronization are described.

1. USART Programming

The 8251A USART converts parallel output data into virtually any serial output data format. The USART also converts serial input data into the parallel data format. Prior to starting to transmit or to receive data, the USART must be loaded with a set of control words. These control words, which define the complete functional operation of the USART, must immediately follow a reset (internal or external). There are two types of control words, namely, a Mode instruction and a Command instruction. Once the Mode instruction has been sent, the Command instruction can be sent at any time prior to a read/write operation. The following assembly code is used to initialize the USART read/write mode:

```
MOV  AL,37H
OUT  ØØDAH,AL
```

During the course of execution, the serial I/O interface routine executes the following assembly code to read the USART status and to read the receiver buffer if any character has been received:

```
IN   AL,ØØDAH
AND  AL,Ø2H
JZ   KEYBOARD1
IN   AL,ØØD8H
```

2. PIT Programming

The 8253 PIT has three independent counters. They are Counter Ø, Counter 1, And Counter 2. The input clock frequency is 22.1184 MHz supplied by a crystal oscillator. The input clock frequencies can be adjusted via jumpers. In this project, Counter 1, with the default factory jumper connection (E59-E6Ø), is used. The input frequency is 153.6 KHz. It is chosen to work in mode Ø so that it gives an interrupt on terminal count. The formula:

$$N = TC$$

where

N is the count value for the counter

T is the desired interrupt time interval in seconds

C is the internal frequency (Hz)

From the above formula, the count number for the counter 1 is found to be 384ØØ decimal (96ØØ hex). Since it is not possible to express this number in four decimal digits, the binary count mode is selected. The counter is initialized by sending a mode control word, followed by a down-count number. Once the counter is initialized, sending the down-count number resets it to the start condition. The following sequence of code is used to program the PIT:

```
MOV  AL,50H
OUT  00D6H,AL
MOV  AL,00H
OUT  00D2H,AL
MOV  AL,60H
OUT  00D6H,AL
MOV  AL,96H
OUT  00D2H,AL
```

3. PIC Programming

The 8259A PIC is programmed in the nested mode. The master PIC with no slaves is accepted. For this particular situation, the initialization words 1,2, and 4 are sent. The initialization word 2 is set to represent the interrupt vector address. This is the address that the control is given when the interrupt 1 occurs (04H). An interrupt mask byte is used to mask out the irrelevant interrupts for the purpose of this thesis. The PIC can be reset after each interrupt simply by sending the EOI (end of interrupt) status byte to the appropriate address. The initialization sequence is as follows:

```
CLI
MOV  AL,13H
OUT  00C0H,AL
MOV  AL,20H
OUT  00C2H
STI
MOV  AL,0FDH
OUT  00C2H,AL
```

To reset the PIC,

```
MOV  AL,20H
OUT  00C0,AL
```

E. ASSEMBLY, COMPILING AND LINKING

The assembly language code was written in RASM-86 and assembled by using the RASM-86 Assembler. This assembler produces relocatable files that can then be linked with other separately compiled or assembled object files by Link 86. This linker accepts three types of input files. Those are the object file, library file, and/or an input file. Input files are very useful tools in that they include input command lines by an input file instead of writing the command line each time the programs are to be linked. The PL/I-86 compiler is used to compile PL/I programs. This compiler requires a 128 Kbyte RAM, as opposed to the PL/I-80 compiler, which requires 48K RAM.

F. TESTING

The hierarchical simulation program modules were designed and tested in a top-down manner. An extensive dynamic debugging module was used in the testing of individual modules (see Appendix F). The testing phase is first started with writing a skeletal model for the real time executive module. The PL/I-86 output statements (stubs), which printed some appropriate numbers, were inserted in the places where the simulation system module programs were invoked. The DDT86 (Dynamic Debugging Tool) was used to test and debug this skeleton program. Since it was a real-time interrupt driven program, the interrupt

enable bit (I) of the program status word was reset to zero (Ø) to control the flow of the program. Some error checkings are inserted in the main assembly language program, ARBITER.A86, to see whether the process switching was done correctly, or to see if any of the stacks allocated to the synchronization primitives have overflowed.

After testing the real-time executive model, the second phase was the testing of the subroutines that are used by the simulation system module programs. Appendix G shows an example for testing a subroutine.

The third phase of the testing was to test and debug the simulation system program. The dynamic debugging module was used for this purpose. This module is composed of PL/I-86 %INCLUDE files and external PL/I-86 programs. The %INCLUDE files are inserted into the various parts of the program being tested. The code of this debugging system is bordered with comment lines from the main body of the program it tests. It is not visible to the program, i.e., it brings its local and global variables with the LOCALS.AID declaration %INCLUDE file. It is possible to manipulate the system's external variables through the debugging system module, IDLE.PLI.

The PL/I-8Ø had been used in the early stages of the testing phases because of the existence of redundant Intel 8Ø8Ø based systems in the Naval Postgraduate Micro-Lab.

But in the later stages, it was realized that this had caused some problems, due to some incompatibilities between the PL/I-80 and PL/I-86 systems. These deficiencies of the PL/I-86 have necessitated the testing and debugging of the programs to be done in the PL/I-86 based systems, The need for such a thorough debugging system module was then realized.

IV. CONCLUSIONS

The original objectives of the thesis have been accomplished, to a large extent. The hardware interconnections intended to promote real time clock operations have been successful. A test model, which comprised the real time executive module and the PL/I put statements, instead of the simulation system modules, was developed to check the system's operation and timing. Correct results appeared on the screen. Then, the testing and debugging of the simulation system programs showed that their algorithms and operations were correct, with the exception of the simulation system program, DISPLAY.PLI. The testing of DISPLAY.PLI showed that the objects to be displayed on the video screen were not successfully put on the screen. One error, which was an automatic conversion error, was found in the routine TRANSLATE and corrected by using a step variable. However, an error still exists in the body of the procedure DRAW.

Since the testing phase of the DISPLAY.PLI has not been accomplished, the objective of synchronizing two iSBC 86/12A boards could not be accomplished. The intended purpose for this objective was to make the second iSBC 86/12A jump to a waiting loop with the initial power start up interrupt (reset), to load the assigned simulation program segment to the common RAM, and to direct it to the

beginning of that program segment, through a bus vectored interrupt.

A hierarchical and modular program model was constructed through the use of data structures and structured programming. The modularization allows possible future changes to the programs.

The simulation system program constructed in this thesis could be a basis for further enhancements towards a complete fire control system, or a related tactical simulation system, due to its modularized nature.

The dynamic debugging module designed to test and debug the simulation system modules can be used for the purpose of testing any other programs, simply by changing the names and formats of the variables that the debugging system manipulates. The advantage of using the %REPLACE and %INCLUDE pre-processor statements, which are peculiar to the PL/I-86 version of the subset G, makes such an implementation feasible.

APPENDIX A

HEAD MODULE PROGRAM LISTINGS

A. WAR.PLI

```
/*
Prog  Name       :  WAR.PLI
Date             :  December 83
Written  by      :  M. Kadri Ozyurt
For              :  Thesis
Advisor          :  Professor Kodres
Purpose          :  This  is  the main  procedure  of  the
modular simulation program.  It invokes the initialization
module to set up the target database and to initialize the
external variables used throughout the simulation program.
*/


WAR:PROCEDURE OPTIONS(MAIN);


/*external procedures*/

  DCL
          (INITVARS,ARBITER) ENTRY;


/*this call to the initialization module initializes the
simulation system*/

CALL INITVARS;

/*this  call  gives  the  control over  to  the  real  time
executive module*/

CALL ARBITER;


END WAR;
```

B. GLOBALS.INP

```
/*
Prog  Name          : GLOBALS.INP
Date                : December 83
Written by          : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This %include file contains the  declara-
tions of the global variables used in the simulation program
*/


  DCL
     (COUNTER,SECONDS,MINUTES,HOURS,WAKE_PTR,
     SHIP_PTR,AVAILSHIP,AVAILWAKE,P,Q,NUMBERSHIPS,
      NODE,TARGET,KNOWN) FIXED BIN(7) EXTERNAL,
      FOURTHEVC FIXED BIN(15) EXTERNAL ,

      D FIXED BIN(7) EXTERNAL,
      (DT,T_PRIME,T_OF,T) FLOAT EXTERNAL,
      (I,J,XX,YY) FIXED BIN(15),
      CURRENTPROC FIXED BIN(7) EXTERNAL ,

      (ENGAGED,MAGNIFIED,FIRED,ERRORON,DONE) BIT(1) EXTERNAL ,

      KEY CHARACTER (1) EXTERNAL ,
      THRESHOLD(0:2) FIXED BIN(15) EXTERNAL ,
      ARG(0:5) FIXED BIN(7) EXTERNAL,

     (VX_OWN,VY_OWN,VX_TARGET,VY_TARGET,VX_REL,
      VY_REL,VX_ROUND,VY_ROUND,VR) FIXED DECIMAL   EXTERNAL,

     ALPHA FIXED DECIMAL  EXTERNAL ,

     (AX_SUM,BX_SUM,CX_SUM,AY_SUM,BY_SUM,CY_SUM,
      AX,BX,CX,AY,BY,CY,X_AT5,Y_AT5,R,DX_DT_AT5,
      DY_DT_AT5,DR_DT_AT5,X_OFFSET,Y_OFFSET,M)FIXED DECIMAL
                                                 EXTERNAL,

      (O1,O2)(5) FIXED DECIMAL  EXTERNAL ,

      1 SHIP(MAX_SHIPS) EXTERNAL,
          2 VELOCITY,
              3 COURSE FIXED DECIMAL   INIT(0.0),
              3 SPEED FIXED DECIMAL   INIT(0.0),
          2 POSITION,
              3 AZIMUTH FIXED DECIMAL   INIT(0.0),
              3 RANGE FIXED DECIMAL   INIT(0.0),
```

```
    2 COORDINATES,
       3 X FIXED DECIMAL   INIT(0.0),
       3 Y FIXED DECIMAL   INIT(0.0),
    2 AIM,
       3 X_AIM FIXED DECIMAL   INIT(0.0),
       3 Y_AIM FIXED DECIMAL   INIT(0.0),
    2 COUNT FIXED BIN (7) INIT(0),
    2 NUMBER FIXED BIN (7) INIT(0),
    2 PTR FIXED BIN (7) INIT(0),
    2 LINK_SHIP FIXED BIN (7) INIT(0),
    2 FRIEND BIT(1) INIT(FALSE),

 1 OBJECT(MAX_SHIPS) EXTERNAL,
    2 LOCATIONS,
       3 U (0:10) FIXED BIN(15) INIT((11) -1),
       3 V (0:10) FIXED BIN(15) INIT((11) -1),
    2 AIMS,
       3 U_AIM (0:10) FIXED BIN(15) INIT((11) -1),
       3 V_AIM (0:10) FIXED BIN(15) INIT((11) -1),
    2 GUN,
       3 U_GUN (0:10) FIXED BIN(15) INIT((11) -1),
       3 V_GUN (0:10) FIXED BIN(15) INIT((11) -1),
    2 WAKES,
       3 U_WAKE (0:10) FIXED BIN(15) INIT((11) -1),
       3 UU_WAKE (0:10) FIXED BIN(15) INIT((11) -1),
       3 V_WAKE (0:10) FIXED BIN(15) INIT((11) -1),
       3 VV_WAKE (0:10) FIXED BIN(15) INIT((11) -1),

 1 GUN EXTERNAL,
    2 POSITION,
       3 AZ FIXED DECIMAL   INIT(0.0),
       3 ALT FIXED DECIMAL   INIT(0.0),
    2 COORDINATES,
       3 X_GUN FIXED DECIMAL   INIT(0.0),
       3 Y_GUN FIXED DECIMAL   INIT(0.0),

 1 WAKE(4) EXTERNAL,
    2 COORDINATES,
       3 X_WAKE FIXED DECIMAL   INIT(0.0),
       3 Y_WAKE FIXED DECIMAL   INIT(0.0),
    2 LINK_WAKE FIXED BIN(7) INIT(0),

 (SUSPEND,RESUME,ARBITER,INITVARS) ENTRY,
 KEYBOARD ENTRY (CHARACTER(1)),
 ATOD ENTRY (FIXED BIN(7),FIXED BIN(7));
```

C. CONST.INP

```
/*
Prog Name       : CONST.INP
Date            : December 83
For             : Thesis
Advisor         : Professor Kodres
Purpose         : This %include file contains the  constant
declarations  used throughout the program.
*/


  %REPLACE
   MAX_WAKE BY 4,                /*max number of wake nodes*/
   MAX_SHIPS BY 2,               /*max number of ship nodes*/
   MAXVARS BY 82,                /*max number of variables*/
   OWN BY 1,                     /*ownship indicator*/
   RMAX BY 25000.0,              /*max gun range*/
   MAXSQ   BY  1.073E+09,        /*max  argument for SQRT */
   TOP BY 32767.0,               /*max number for fixed (15)*/
   G BY 10.7246,                 /*gravitational con.yd/sec2*/
   VM BY 518.0,                  /*muzzle velocity*/
   A BY 512,                     /*x coord. for center*/
   B BY 390,                     /*y coord. for center*/
   K BY 1.40625,                 /*azimuth proportionality c.*/
   L BY 4.513,                   /*speed proportionality c.*/
   TWO_PI BY 360.0,              /*definition for 360 degree*/
   PI BY 3.1416,                 /*definition of pi rad/180deg*/
   O0 BY 1.0,                    /*legendre poly. of zero deg.*/
   TRUE BY '1'B,                 /*boolean true*/
   FALSE BY '0'B,                /*boolean false*/
   NIL BY 0,                     /*in linked list terminology*/
   CLEAR_SCREEN BY '^\^[^L',     /*char. sequence for retro.*/
   CLEAR_ALPHA BY '^Z',          /*  "     "      "    "    */
   VECMOD BY '^]',               /*  "     "      "    "    */
   POINTMOD BY '^\',             /*  "     "      "    "    */
   ALPHA4010 BY '^\^ ',          /*  "     "      "    "    */
   ALPHA3A BY '^\^_^X',          /*  "     "      "    "    */
   WHITEMOD BY '^[a',            /*  "     "      "    "    */
   BLACKMOD BY '^[';             /*  "     "      "    "    */
```

APPENDIX B

INITIALIZATION MODULE PROGRAM LISTINGS

A. INITVARS.PLI

```
/*
Prog Name           : INITVARS.PLI
Date                : December 83
Written by          : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This routine prompts the user to give
the time interval (dt) and constructs the tactical circular
linked  list  in an interactive manner.  The  control  then
proceeds to initialize the external variables.
*/

initvars:proc external;

/*
  dcl
*/
         %include'const.inp';
         %include'globals.inp';


  /*this iterative loop initializes the pool of available
wake nodes*/
  do i=1 to max_wake-1;
         link_wake(i)=i+1;
  end;
  link_wake(max_wake)=nil;

/*this  sequence  initializes  the pool of  available  ship
nodes*/
  do i=1 to max_ships-1;
         link_ship(i)=i+1;
  end;
  link_ship(max_ships)=nil;
  put skip list('Enter the time interval (dt) in seconds');
```

```
 /* the following block is an interceptor for a too large
input value*/
  on overflow begin;
          put list('*** too large, try again');
          goto init1;
  end;
init1:
  put skip list('>');
  get list(dt);
  revert overflow;
  put skip list('Construction of the tactical database');
  numberships=0;
  done=false;

/*  this procedure call gets a ship node from the  pool  of
available ships and assigns its address to ship pointer*/
  ship_ptr=getship();
  do while (~done);
          put skip list('Enter the position of ',numberships,
                  'th ship in true azimuth and in yards');
/*the  following  two on condition bodies are for the  input
line at init2*/
          on error begin;
                  put list('*** bad value,try again');
                  goto init2;
          end;
          on fixedoverflow begin;
                  put list('*** too large,try again');
                  goto init2;
          end;
init2:
          put skip list('>');
          get list(azimuth(ship_ptr),range(ship_ptr));
          revert error;
          revert fixedoverflow;
          put skip list('Friend or foe (F/E)?');
          put skip list('>');
          get list(key);

/*  the  following  sequence  adds the  ship  node  to  the
appropriate  circular  linked list,  friend ships or  enemy
ships, according to the friend boolean value entered*/
          if (key='F')!(key='f') then do;
              friend(ship_ptr)=true;
              if link_ship(own)=nil then
                  link_ship(ship_ptr)=ship_ptr;
              else
                  link_ship(ship_ptr)=link_ship(own);
              /*end if*/
```

```
            link_ship(own)=ship_ptr;
            end /*do*/;
        else do;
             friend(ship_ptr)=false;
             if ptr(own)=nil then
                link_ship(ship_ptr)=ship_ptr;
            else
                link_ship(ship_ptr)=ptr(own);
             /*end if*/
             ptr(own)=ship_ptr;
        end /*if*/;

/* the following procedure call and the iterative loop get
available nodes from the pool,  construct a circular linked
list of four nodes,  and assign  the address of the list to
the   ptr  pointer   of   the  ship   node   which   is   being
constructed*/
        wake_ptr=getwake();
        q=wake_ptr;
        do i=1 to 4;
                p=wake_ptr;
                wake_ptr=getwake();
                link_wake(p)=wake_ptr;
        end /*do*/;
        link_wake(wake_ptr)=q;
        ptr(ship_ptr)=wake_ptr;
        number(ship_ptr)=numberships;
        put skip list('Would you like to enter another ',
                        'ship (Y/N)?');
        put skip list('>');
        get list(key);
        if (key='Y')!(key='y') then do;
                ship_ptr=getship();
                if ship_ptr=nil then
                        done=true;
                /*end if*/
                end;
        else
                done=true;
        /*end if*/
  end /*do*/;
  seconds=0;
  minutes=0;
  hours=0;
  wake_ptr=0;
  target=0;
  t_prime=0.0;
  t_of=0.0;
  t=0.0;
```

```
  friend(own)=true;
  known=2;                  /* engaged ship no by the sensor */
  x_offset=0.0;
  y_offset=0.0;
  m=50.0;
  fourthevc=0;
  i=0;
  j=0;
  currentproc=4;
  engaged=false;
  magnified=false;
  fired=false;
  erroron=false;
  key='0';
  threshold(0)=1; threshold(1)=4; threshold(2)=4;
  vx_own=0.0;
  vy_own=0.0;
  vx_target=0.0;
  vy_target=0.0;
  vx_rel=0.0;
  vy_rel=0.0;
  vx_round=0.0;
  vy_round=0.0;
  vr=0.0;
  alpha=0.0;
  ax_sum=0.0; bx_sum=0.0; cx_sum=0.0;
  ay_sum=0.0; by_sum=0.0; cy_sum=0.0;
  ax=0.0;bx=0.0;cx=0.0;
  ay=0.0;by=0.0;cy=0.0;
  x_at5=0.0;y_at5=0.0;
  r=0.0;
  dx_dt_at5=0.0; dy_dt_at5=0.0; dr_dt_at5=0.0;
  o1(1)=1.0; o1(2)=0.5; o1(3)=0.0; o1(4)=-0.5; o1(5)=-1.0;
  o2(1)=1.0; o2(2)=-0.5; o2(3)=-1.0; o2(4)=-0.5; o2(5)=1.0;

/*getship, when invoked, extracts a node from the pool of
available and returns a pointer value pointing to that
node. It puts an error message if there is no available
node*/
getship:procedure returns (fixed bin(7));

  /* dcl */
        %include 'globals.inp';
  if availship=nil then do;
        put skip list('No more available ship nodes');
        return(nil);
        end /*do*/;
  else do;
        node=availship;
        availship=link_ship(availship);
  end /*if*/;
```

```
  return(node);
end getship;


/*getwake  does the same function as getship except for the
operations made are on wake nodes*/
getwake:procedure returns (fixed bin(7));

  /* dcl */
        %include 'globals.inp';

  if availwake=nil then do;
        put skip list('No more available wake nodes');
        return (0);
        end /*do*/;
  else do;
        node=availwake;
        availwake=link_wake(availwake);
  end /*if*/;
  return(node);
end getwake;

end initvars;
```

APPENDIX C

SIMULATION SYSTEM MODULE PROGRAM LISTINGS

A. TACTICAL.PLI

```
/*
Prog Name             : TACTICAL.PLI
Date                  : December 83
Written by            : M. Kadri Ozyurt
For                   : Thesis
Advisor               : Professor Kodres
Purpose               : This external routine calculates  and
updates  the positions of the ships in the  tactical  area
and  the  future  positions of  the ships that belong  to
enemy  ships circular linked list pointed to  by  ptr(own),
and
and  calculates the trajectory of the travelling projectile
if fired
*/

TACTICAL:PROCEDURE EXTERNAL;




/*
  DCL
*/
    %INCLUDE 'CONST.INP';
    %INCLUDE 'GLOBALS.INP';



/*following sequence of code updates the present positions
of the ships in the tactical area*/
  DO I=2 TO NUMBERSHIPS;
      VX_TARGET = SPEED(I) * SIND (COURSE(I));
      VY_TARGET = SPEED(I) * COSD (COURSE(I));
      VX_REL = VX_TARGET - VX_OWN;
      VY_REL = VY_TARGET - VY_OWN;
      WAKE_PTR = PTR(I);
```

```
    X_WAKE(WAKE_PTR) = X(I);
    Y_WAKE(WAKE_PTR) = Y(I);
    X(I) = X(I) + VX_REL * DT;
    Y(I) = Y(I) + VY_REL * DT;
    PTR(I) = LINK_WAKE(WAKE_PTR);  /*ptr(i) points to*/
    IF COUNT(I) <~4 THEN            /* the oldest wake*/
        COUNT(I)=COUNT(I)+1;
    /*END IF*/
  END /*DO*/;


/*calculating the future positions (aim points) starts
here by using the least squares method with legendre
polynomials. The coefficients are pre-calculated according
to the fifth second including the zeroth second*/
FILTERING:
  TARGET = PTR(OWN);
  DONE=TRUE;
  IF ~(TARGET=NIL) THEN
        DONE=FALSE;
  /*END IF*/
  DO WHILE (~DONE);
      IF COUNT(TARGET)=4 THEN
        BEGIN;
          AX_SUM = X(TARGET) * 00;    /*Leg. poly. 0 deg.*/
          BX_SUM = X(TARGET) * 01(5);/*  "      "   1  "  */
          CX_SUM = X(TARGET) * 02(5);/*  "      "   2  "  */
          AY_SUM = Y(TARGET) * 00;
          BY_SUM = Y(TARGET) * 01(5);
          CY_SUM = Y(TARGET) * 02(5);
          WAKE_PTR = PTR(TARGET);
          J = 1;
          DO WHILE (LINK_WAKE(WAKE_PTR) ~= PTR(TARGET));
              AX_SUM = AX_SUM + X_WAKE(WAKE_PTR) * 00;
              BX_SUM = BX_SUM + X_WAKE(WAKE_PTR) * 01(J);
              CX_SUM = CX_SUM + X_WAKE(WAKE_PTR) * 02(J);
              AY_SUM = AY_SUM + Y_WAKE(WAKE_PTR) * 00;
              BY_SUM = BY_SUM + Y_WAKE(WAKE_PTR) * 01(J);
              CY_SUM = CY_SUM + Y_WAKE(WAKE_PTR) * 02(J);
              WAKE_PTR=LINK_WAKE(WAKE_PTR);
              J = J + 1;
          END /*DO*/;

          AX = AX_SUM / 5.;
          BX = 2.0 * BX_SUM / 5.;
          CX = 2.0 * CX_SUM / 7.;
          AY = AY_SUM / 5.;
          BY = 2.0 * BY_SUM / 5.;
          CY = 2.0 * CY_SUM / 7.;
```

```
        X_AT5 = AX - 1.5*BX + 3.5*CX;
        Y_AT5 = AY - 1.5*BY + 3.5*CY;


INCONVENIANCE:
/*this  begin block is inserted to avoid the  complications
which might arise from the automatic conversions*/
        BEGIN;
            DCL  (XSTEP,YSTEP) FIXED,
                 (XF,YF,RSQD) FLOAT;

             IF (ABS(X_AT5)>TOP)!(ABS(Y_AT5)>TOP) THEN
                  RSQD=-1.0;
              ELSE DO;
                  XSTEP=BINARY(X_AT5);
                  YSTEP=BINARY(Y_AT5);
                  XF=FLOAT(XSTEP);
                  YF=FLOAT(YSTEP);
                  RSQD=XF*XF+YF*YF;
              END /*IF*/;
             IF (RSQD<0.0)!(RSQD>MAXSQ) THEN
                  R=0.0;
             ELSE
                  R=SQRT(RSQD);
              /*END IF*/
         END INCONVENIANCE;

         IF (R=0.0) ! (R>RMAX) THEN DO;
                  X_AIM(TARGET)=0.0;
                  Y_AIM(TARGET)=0.0;
              END /*DO*/;
         ELSE DO;
                  ALPHA = ASIN(G*R/VM**2) / 2.0;/*IN RADS*/
                  VR = VM * COS(ALPHA);
                  T_PRIME = R / VR;
                  DX_DT_AT5 = 3.0*CX - 0.5*BX;
                  DY_DT_AT5 = 3.0*CY - 0.5*BY;
                  DR_DT_AT5 = (X_AT5 * DX_DT_AT5 +
                                  Y_AT5 * DY_DT_AT5) / R;
                  T_OF = (R + DR_DT_AT5 * T_PRIME) / VR;
                  X_AIM(TARGET) = X_AT5 + DX_DT_AT5 * T_OF;
                  Y_AIM(TARGET) = Y_AT5 + DY_DT_AT5 * T_OF;
         END /*IF*/;
     END /*IF*/;
     TARGET=LINK_SHIP(TARGET);           /*next target?*/
     IF TARGET=PTR(OWN) THEN
         DONE=TRUE;
     /*END IF*/
 END /*DO*/;
```

```
ROUNDTRACK:
/*the ballistic calculations start here*/
  IF (FIRED) THEN
     BEGIN;
          VX_REL = VX_ROUND - VX_OWN;
          VY_REL = VY_ROUND - VY_OWN;
          X_GUN = X_GUN + VX_REL * DT;
          Y_GUN = Y_GUN + VY_REL * DT;
          T = T - DT;
          IF T<=0 THEN DO;
               PUT LIST('^G');
              FIRED = FALSE;
          END /*IF*/;
  END /*IF*/;


END TACTICAL;
```

B. DISPLAY.PLI

```
/*
Prog  Name          : DISPLAY.PLI
Date                : December 83
Written by          : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose              :  This  routine first puts the time  in
hours, minutes, and seconds. The control then proceeds to
call DRAW subroutine in an iterative loop to erase the old
objects. It then calculates the  positions of the  objects
relative  to  either  ownship  or the ship  that  has  been
targeted  according  to  MAGNIFIED.Then it  calls  DRAW  to
display the objects.
*/




DISPLAY:PROCEDURE EXTERNAL;

/*DCL*/
      %INCLUDE 'CONST.INP';
      %INCLUDE 'GLOBALS.INP';


  PUT LIST('^]^_^X');        /*ENTER ALPHA MODE*/
  IF MINUTES=0 THEN
      BEGIN;
          PUT LIST ('^[= %');
          PUT EDIT (HOURS)(F(2));
          HOURS = HOURS+1;
          IF HOURS=24 THEN
             HOURS = 0;
          /*END IF*/
  END /*IF*/;

  IF SECONDS=0 THEN
      BEGIN;
          PUT LIST ('^[= (');
          PUT EDIT (MINUTES)(F(2));
          MINUTES = MINUTES+1;
          IF MINUTES=60 THEN
             MINUTES = 0;
          /*END IF*/
  END /*IF*/;

  PUT LIST ('^[= +');
  PUT EDIT (SECONDS)(F(2));
```

```
  SECONDS = SECONDS+1;
  IF SECONDS=60 THEN
      SECONDS = 0;
  /*END IF*/
  PUT LIST('^M^X^^');                /*HOME CURSOR*/

/*the following calls erase the objects from the screen*/
  D=0;
  DO I=1 TO NUMBERSHIPS;
          CALL DRAW(OBJECT(I).U,OBJECT(I).V,D);
          CALL DRAW(OBJECT(I).UU_WAKE,OBJECT(I).VV_WAKE,D);
          CALL DRAW(OBJECT(I).U_AIM,OBJECT(I).V_AIM,D);
  END /*DO*/;
  CALL DRAW(OBJECT(1).U_GUN,OBJECT(1).V_GUN,D);

/*the  following sequence converts the coordinates  of  the
objects to the grid coordinates of the screen and generates
the sequences of the coordinates for the objects*/
 DO I=1 TO NUMBERSHIPS;
         XX=A+BINARY((X(I)-X_OFFSET)/M);
         YY=B+BINARY((Y(I)-Y_OFFSET)/M);
         IF (FRIEND(I)) THEN
            CALL GENFRIEND(XX,YY,OBJECT(I).U,OBJECT(I).V);
         ELSE
            CALL GENFOE(XX,YY,OBJECT(I).U,OBJECT(I).V);
         /*END IF*/
         CALL GENWAKE(XX,YY,OBJECT(I).U_WAKE,OBJECT(I).V_WAKE);
         XX=A+BINARY((X_WAKE(PTR(I))-X_OFFSET)/M);
         YY=B+BINARY((Y_WAKE(PTR(I))-Y_OFFSET)/M);
         IF COUNT(I)=4 THEN
            CALL GENWAKE(XX,YY,OBJECT(I).UU_WAKE,
                                       OBJECT(I).VV_WAKE);
         XX=A+BINARY((X_AIM(I)-X_OFFSET)/M);
         YY=B+BINARY((Y_AIM(I)-Y_OFFSET)/M);
         IF ^((XX=A)&(YY=B)) THEN
              IF I=1 THEN
                 CALL GENOURAIM(XX,YY,OBJECT(I).U_AIM,
                                          OBJECT(I).V_AIM);
              ELSE
                 CALL GENAIM(XX,YY,OBJECT(I).U_AIM,
                                         OBJECT(I).V_AIM);
              /*END IF*/
         /*END IF*/
  END /*DO*/;
  XX=A+BINARY((X_GUN-X_OFFSET)/M);
  YY=B+BINARY((Y_GUN-Y_OFFSET)/M);
  CALL GENGUN(XX,YY,OBJECT(1).U_GUN,OBJECT(1).V_GUN);
```

```
/*the  following sequence draws the objects by calling  the
routine DRAW*/
  D=1;
  DO I=1 TO NUMBERSHIPS;
      CALL DRAW(OBJECT(I).U,OBJECT(I).V,D);
      CALL DRAW(OBJECT(I).U_WAKE,OBJECT(I).V_WAKE,D);
      IF (ENGAGED) THEN CALL DRAW(OBJECT(I).U_AIM,
                                  OBJECT(I).V_AIM,D);
  END /*DO*/;
  IF FIRED THEN CALL DRAW(OBJECT(1).U_GUN,OBJECT(1).V_GUN,D);

/*  the following procedures produce the sequence of screen
grid coordinates for various objects*/

GENFRIEND:PROC(X,Y,U,V);

  DCL
          (X,Y) FIXED BIN(15),
          (U,V)(Ø:1Ø) FIXED BIN(15);

  U(Ø)=X;          V(Ø)=Y-8;
  U(1)=X-6;        V(1)=Y-3;
  U(2)=X-6;        V(2)=Y+3;
  U(3)=X;          V(3)=Y+8;
  U(4)=X+6;        V(4)=Y+3;
  U(5)=X+6;        V(5)=Y-3;
  U(6)=X;          V(6)=Y-8;
  U(7)=-1;         V(7)=-1;

END GENFRIEND;

GENFOE:PROC(X,Y,U,V);

  DCL
          (X,Y) FIXED BIN(15),
          (U,V)(Ø:1Ø) FIXED BIN(15);

  U(Ø)=X+8;        V(Ø)=Y-4;
  U(1)=X-8;        V(1)=Y-4;
  U(2)=X;          V(2)=Y+8;
  U(3)=X+8;        V(3)=Y-4;
  U(4)=-1;         V(4)=-1;

END GENFOE;
```

```
GENWAKE:PROC(X,Y,U,V);

  DCL
          (X,Y) FIXED BIN(15),
          (U,V)(Ø:1Ø) FIXED BIN(15);

  U(Ø)=X;          V(Ø)=Y;
  U(1)=X;          V(1)=Y;
  U(2)=-1;         V(2)=-1;

END GENWAKE;

GENOURAIM:PROC(X,Y,U,V);

  DCL
          (X,Y) FIXED BIN(15),
          (U,V)(Ø:1Ø) FIXED BIN(15);

  U(Ø)=X+8;        V(Ø)=Y;
  U(1)=X-8;        V(1)=Y;
  U(2)=X;          V(2)=Y;
  U(3)=X;          V(3)=Y-8;
  U(4)=X;          V(4)=Y+8;
  U(5)=-1;         V(5)=-1;

END GENOURAIM;

GENAIM:PROC(X,Y,U,V);

  DCL
          (X,Y) FIXED BIN(15),
          (U,V)(Ø:1Ø) FIXED BIN(15);

  U(Ø)=X+4;        V(Ø)=Y+4;
  U(1)=X-4;        V(1)=Y-4;
  U(2)=X;          V(2)=Y;
  U(3)=X+4;        V(3)=Y-4;
  U(4)=X-4;        V(4)=Y+4;
  U(5)=-1;         V(5)=-1;

END GENAIM;
```

```
GENGUN:PROC(X,Y,U,V);

  DCL
          (X,Y) FIXED BIN(15),
          (U,V)(Ø:1Ø) FIXED BIN(15);

  U(Ø)=X+1;        V(Ø)=Y-1;
  U(1)=X+1;        V(1)=Y+1;
  U(2)=X-1;        V(2)=Y+1;
  U(3)=X-1;        V(3)=Y-1;
  U(4)=X+1;        V(4)=Y-1;
  U(5)=-1;         V(5)=-1;

END GENGUN;


/*this procedure receives two arrays and a key variable  as
paramaters, and either displays the object or erases it*/
DRAW:PROC(U,V,D) EXTERNAL;

  DCL
          (U,V)(Ø:1Ø) FIXED BIN(15),
          (I,J,H,D) FIXED BIN(7),
          RUB CHAR(1) EXTERNAL,
          C(7) CHAR(1),
          Z1(Ø:3) CHAR(1) BASED(P),
          Z(Ø:3) BIT(8),
          P POINTER;

  P=ADDR(Z);        /*Z and Z1 share same location hereon*/
  I=Ø;
  DO WHILE (I<11);
          IF (D=1)
              /*enter vector set level wnite*/
              THEN PUT LIST('^]^[a');
              /*enter vector set level black*/
              ELSE PUT LIST('^]^[',RUB);
          /*END IF*/
          DO H=1 TO 5;
                  IF (U(I)<Ø) THEN DO;
                          PUT LIST('^M^X^[=  ');
                          RETURN;
                  END /*IF*/;
            /*this call translates the coordinates to the
              stream of bits*/
                  CALL TRANSLATE(U(I),V(I),Z);
                  I=I+1;
```

```
            /*this put statement puts out the bit streams
              as being characters*/
                 PUT EDIT((Z1(J) DO J=0 TO 3)) (4A(1));
         END /*DO*/;
      /*the  following  two statements get the status  of
      the screen.  The status is not sent back until  the
      screen is ready.*/
         PUT LIST('^[^E');        /*HANDSHAKE*/
         GET EDIT((C(J) DO J=1 TO 7))(7A(1));
  END /*DO*/;
  PUT LIST('^M^X');          /*BACK TO ALPHA*/

TRANSLATE:PROCEDURE(X,Y,Z) ;
  DECLARE (X,Y) FIXED BIN(15),
          Z(0:3) BIT(8),
          T FIXED BIN(15),
          S1 BIT (16),
          SS BIT(7),
          S BIT (8),
          I FIXED BIN(7);
  I = DIVIDE(Y,32,8);
  SS=BIT(I,7);
  S='0'B || SS;
  Z(0) = '00100000'B ! S;
  T = Y - (Y/32) * 32;
  S1 = BIT (T,16);
  S = SUBSTR(S1,8,8);
  Z(1) = '01100000'B ! S;
  I = DIVIDE (X,32,8);
  SS=BIT(I,7);
  S='0'B || SS;
  Z(2) = '00100000'B ! S;
  T = X - (X/32) * 32;
  S1 = BIT(T,16);
  S = SUBSTR (S1,8,8);
  Z(3) = '01000000'B ! S;
 END TRANSLATE;

 END DRAW;

END DISPLAY;
```

C. STATUS.PLI

```
/*
Prog  Name          : STATUS.PLI
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : this routine calls the assembly routine
KEYBOARD  to read the keyboard to set the boolean variables
used in other routines
*/




STATUS:PROCEDURE EXTERNAL;




/*
  DCL
*/
     %INCLUDE 'CONST.INP';
     %INCLUDE 'GLOBALS.INP';


      CALL KEYBOARD(KEY);
      IF (KEY='Q')!(KEY='q') THEN
          STOP;
      ELSE IF (KEY='E')!(KEY='e') THEN
          ENGAGED = TRUE;
      ELSE IF ENGAGED & ((RANK(KEY)>48)&(RANK(KEY)<=57))
        THEN BEGIN;
          I=RANK(KEY)-48;
          IF NUMBER(I)=Ø THEN
                  LINK_SHIP(I)=I;
          ELSE IF FRIEND(I) THEN DO;
                  CALL REMOVENODE(LINK_SHIP(OWN),I);
                  CALL ADDNODE(PTR(OWN),I);
                  FRIEND(I)=FALSE;
                  END;
          PTR(OWN) = I;
          END;
      ELSE IF (KEY='R') ! (KEY='r') THEN
          ENGAGED=FALSE;
      ELSE IF (KEY='M')!(KEY='m') THEN DO;
```

```
        /*set the scale tc 1/200*/
           MAGNIFIED=TRUE;
           M=200.0;
           IF ENGAGED THEN DO;
           /*set reference as the target*/
                   X_OFFSET=X(PTR(OWN));
                   Y_OFFSET=Y(PTR(OWN));
                   END /*DO*/;
           ELSE DO;
           /*ownship is the reference*/
                   X_OFFSET=0.0;
                   Y_OFFSET=0.0;
                   END /*DO*/;
           /*END IF*/
       END;
       ELSE IF (KEY='T')!(KEY='t') THEN DO;
       /*set the scale back to normal (1/50)*/
                   MAGNIFIED=FALSE;
                   M=50.0;
                   X_OFFSET=0.0;
                   Y_OFFSET=0.0;
         END /*DO*/;
       ELSE IF (KEY='F')!(KEY='f') THEN
           FIRED = TRUE;
       ELSE IF (KEY='D')!(KEY='d') THEN
           ERRORON=TRUE;
       ELSE IF (KEY='S')!(KEY='s') THEN
           SIGNAL ERROR(1);
       /*END IF*/
       KEY='0';


/*this  routine  removes the node pointed by  QQ  from  the
circular linked linked list pointed by PP*/
REMOVENODE:PROC(PP,QQ);

  DCL
          (PP,QQ) FIXED BIN(7);
          %INCLUDE'GLOBALS.INP';

  P=PP;
  P=LINK_SHIP(P);
  DO WHILE (^(LINK_SHIP(P)=QQ));
          P=LINK_SHIP(P);
  END;
  LINK_SHIP(P)=LINK_SHIP(QQ);
END;
```

```
/*this  routine  adds a node pointed by QQ to the  circular
linked list by PP*/
ADDNODE:PROC(PP,QQ);

  DCL
          (PP,QQ) FIXED BIN(7);
          %INCLUDE'GLOBALS.INP';

  P=PP;
  P=LINK_SHIP(P);
  DO WHILE (~(LINK_SHIP(P)=PP));
          P=LINK_SHIP(P);
  END;
  LINK_SHIP(P)=QQ;
  LINK_SHIP(QQ)=PP;
END;

END STATUS;
```

D. IDLE.PLI

```
/*
Prog  Name          : IDLE.PLI
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This routine reads the A/D converter
output  to  get  the  velocity vectors  of  ownship  and  a
selected  ship,  and  the gun information  as  azimuth  and
elevation.  It then converts this information to real world
values.  It calculates  ownship speed which will be used to
find  relative speeds later.  It then computes the  maximum
range,  cartesian coordinates of the splash point, and time
of flight corresponding to the current gun position
*/



IDLE:PROCEDURE EXTERNAL;



/*
  DCL
*/
      %INCLUDE 'CONST.INP';
      %INCLUDE 'GLOBALS.INP';

      DO D=Ø TO 5;
          CALL ATOD (D,ARG(D));
      END /*DO*/;

/*at  this  point  the A/D output values are  fixed  bin(7)
values.  The  following  sequence converts those  to  fixed
decimal values*/
      COURSE(OWN)=ARG(Ø);
      SPEED(OWN)=ARG(4);
      COURSE(KNOWN)=ARG(2);
      SPEED(KNOWN)=ARG(3);
      AZ=ARG(1);
      ALT=ARG(5);

/*the  following sequence converts A/D values to real  time
values by using appropriate proportionality constants*/
      COURSE(OWN) = COURSE(OWN) * K;
```

```
      COURSE(KNOWN) = COURSE(KNOWN) * K;
      AZ = AZ * K;
      IF COURSE(OWN)<0.0 THEN
          COURSE(OWN) = COURSE(OWN) + TWO_PI ;
      IF COURSE(KNOWN)<0.0 THEN
          COURSE(KNOWN) = COURSE(KNOWN) + TWO_PI ;
      IF AZ<0.0 THEN
          AZ = AZ + TWO_PI;
       IF ALT>90.0 THEN
          ALT = 90.0;

          SPEED(OWN) = SPEED(OWN)/L;
          SPEED(KNOWN) = SPEED(KNOWN) / L;

/*ownship speed computations*/
      VX_OWN = SPEED(OWN) * SIND(COURSE(OWN));
      VY_OWN = SPEED(OWN) * COSD(COURSE(OWN));

/*when not have fired, the following makes the ballistic
computations*/
      IF ~ FIRED THEN
         BEGIN;
          T_OF = 2.0 * VM * SIND(ALT) / G;
          VR = VM * COSD(ALT);
          R = VR * T_OF;
          X_AIM(OWN) = R * SIND(AZ);
          Y_AIM(OWN) = R * COSD(AZ);
          X_GUN = 0.0;
          Y_GUN = 0.0;
          VX_ROUND = VR * SIND(AZ);
          VY_ROUND = VR * COSD(AZ);
          T = T_OF;
      END /*IF*/;
END IDLE;
```

APPENDIX D

REAL TIME EXECUTIVE MODULE LISTINGS

A. ARBITER.A86

```
;Prog  Name            : ARBITER.A86
;Date                  : December 83
;Written  by           : M. Kadri Ozyurt
;For                   : Thesis
;Advisor               : Professor Kodres
;Purpose               : This program contains all the assembly
;routines used by the simulation system. It initializes all
;programmable  hardware components,  responds to the  timing
;interrupts, and increment the FOURTHEVC used throughout the
;simulation  model  program.  Upon  receiving  interrupt
;requests  ,  it  performs process switching by storing  the
;state  of interrupted process in the stack  area  allocated
;for  the  processes  and  by restoring  the  highest  ready
;process given by SCHEDULER




;GLOBALS

DGROUP    GROUP   FOURTHEVC,CURRENTPROC
FOURTHEVC DSEG    COMMON
FOURTHEVC1 DW     Ø
CURRENTPROC DSEG COMMON
CURRENTPROC1 DB   4

  CSEG
  EXTRN   SCHEDULE:FAR
  EXTRN   P1:FAR
  EXTRN   P2:FAR
  EXTRN   P3:FAR
  EXTRN   P4:FAR
  PUBLIC  ARBITER
  PUBLIC  STORESTATUS
  PUBLIC  RINGBELL
  PUBLIC  KEYBOARD
  PUBLIC  ATOD
  PUBLIC  SUSPEND
  PUBLIC  RESUME
```

```
;EQUATES

  INT1     EQU      84H                           ;INTR1 JUMP ADDR.
  INT3     EQU      8CH                           ;INTR3  JUMP ADDRESS
  PIC1     EQU      0C0H                          ;PIC COMMAND OUTPUT PORT1
  PIC2     EQU      0C2H                          ;PIC COMMAND OUTPUT PORT2
  ICW1     EQU      13H                           ;PIC COMMAND WORD1
  ICW2     EQU      20H                           ;PIC COMMAND WORD2
  ICW4     EQU      0DH                           ;PIC COMMAND WORD4
  MASK1    EQU      0FDH                          ;PIC MASK BYTE
  EOI      EQU      20H                           ;END-OF-INTERRUPT BYTE
  CNTR1    EQU      50H                           ;PIT MODE CONTROL BYTE
  CNTR2    EQU      60H                           ; "   "    "     "
  PORTC    EQU      0D6H                          ;PIT CONTROL PORT
  COUNT    EQU      0D2H                          ;PIT COUNT # OUTPUT PORT
  CNTRLO EQU        00H                           ;PIT COUNT # LO BYTE
  CNTRHI EQU        96H                           ;PIT COUNT # HI BYTE
  READWR   EQU      37H
  RXRDY    EQU      02H                           ;USART STATUS MASK(READ)
  TXRDY    EQU      01H                           ;USART STATUS MASK(WRITE)
  PORTIO EQU        0D8H                          ;USART I/O PORT
  PORTST   EQU      0DAH                          ;USART STATUS PORT
  SEGCONV EQU       0D000H                        ;A/D CONVTR PORT SEGMENT
  OFFCONV EQU       0F700H                        ;A/D CONVTR PORT OFFSET
  STACKSIZE EQU     100H
  LF       EQU      0AH
  CR       EQU      0DH
  BEL      EQU      07H
  FS       EQU      1CH
  ESC      EQU      1BH
  FF       EQU      0CH
  CAN      EQU      18H
  FALSE    EQU      0
  TRUE     EQU      NOT FALSE
;
;
ARBITER:
           PUSH     DS
           CLI                                    ;DISABLE INTR'S
           MOV      AX,0
           MOV      DS,AX                         ;SET SEGREG TO 0
           MOV      BX,INT1
           MOV      WORD PTR [BX],OFFSET PROC0 ;INT1 JMP ADDRESS
           INC      BX
           INC      BX
           MOV      WORD PTR [BX],CS
           POP      DS
           MOV      BX,OFFSET STACKTBL+STACKSIZE-2
           MOV      CS:[BX],CS
           ADD      BX,STACKSIZE
```

```
        MOV     CS:[BX],CS
        ADD     BX,STACKSIZE
        MOV     CS:[BX],CS
        ADD     BX,STACKSIZE
        MOV     CS:[BX],CS
        CLI
        MOV     AL,ICW1                        ;INIT. PIC TO
        OUT     PIC1,AL                        ; EDGE-TRIG., SINGLE PIC
        MOV     AL,ICW2
        OUT     PIC2,AL                        ; INT1 ADDR IS Ø4H
        MOV     AL,ICW4
        OUT     PIC2,AL                        ; NOT F. NESTED, NORM. EOI
        STI
        MOV     AL,MASK1
        OUT     PIC2,AL         ;ONLY INT1 IS ALLOWED
        MOV     AL,CNTR1                       ;INIT. PIT
        OUT     PORTC,AL                       ;SELECT MODE Ø ,CNTR 1
                                               ; CLK FREQ. IS 153.6 KHZ
        MOV     AL,CNTRLO                      ;COUNT-DOWN VALUE 96ØØH
        OUT     COUNT,AL                       ; WHICH GIVES AN INTR AT
        MOV     AL,CNTR2
        OUT     PORTC,AL
        MOV     AL,CNTRHI                      ; EVERY FORTH OF A SEC.
        OUT     COUNT,AL
        MOV     AL,READWR
        OUT     PORTST,AL
        MOV     AX,CS
        MOV     SS,AX                          ;SET STACK SEG. TO CODE
        MOV     BP,3*STACKSIZE
        MOV     SP,STACKTBL [BP]
        JMP     P4

;
PROCØ:
        PUSH    AX
        PUSH    BX
        PUSH    CX
        MOV     AL,CNTR2
        OUT     PORTC,AL                       ;RESET COUNTER
        MOV     AL,CNTRHI                      ;RESET THE CNTR.
        OUT     COUNT,AL
        MOV     AL,EOI                         ;RESET PIC
        OUT     PIC1,AL
        ADD     SP,6                           ;SP->INTERRUPTED IP
        POP     BX                             ;BX->      "       IP
        POP     AX                             ;AX->      "        CS
        POP     CX                             ;SP->INTERRUPTED IP
        PUSH    AX                             ;PUSH      "        CS
        PUSH    BX                             ;PUSH      "        IP
        PUSH    CX
```

```
        SUB     SP,6                            ;SP->PUSHED CX
        POP     CX
        POP     BX                              ;RESTORE BX
        POP     AX                              ;RESTORE AX
        INC     FOURTHEVC1
        JMP     STORESTATUS1
STORESTATUS:
        DEC     SP
        DEC     SP
        PUSH    AX
        ADD     SP,4
        POP     AX                              ;AX->INTERRUPTED IP
        PUSH    CS
        PUSH    AX
        DEC     SP
        DEC     SP
        POP     AX                              ;RESTORE AX
        PUSHF
        CLI
STORESTATUS1:
        PUSH AX ! PUSH BX ! PUSH CX ! PUSH DX
        PUSH BP ! PUSH SI ! PUSH DI ! PUSH ES
        CALL    SCHEDULE
        MOV     NEWPROC,AL
        MOV     AL,CURRENTPROC1
        CMP     AL,1
        JNZ     OUT1
        MOV     BP,1
        JMP     OUT4
OUT1:   CMP     AL,2
        JNZ     OUT2
        MOV     BP,STACKSIZE+1
        JMP     OUT4
OUT2:   CMP     AL,3
        JNZ     OUT3
        MOV     BP,2*STACKSIZE+1
        JMP     OUT4
OUT3:   CMP     AL,4
        JNZ     OUT5
        MOV     BP,3*STACKSIZE+1
        JMP     OUT4
OUT4:   ADD     BP,OFFSET STACKTBL
        CMP     SP,BP
        JNA     OUT6
        DEC     BP
        MOV      [BP],SP
        JMP     LOADPROC
OUT5:   ADD     AL,30H
        MOV     OUT5AL,AL
        MOV     DX,OFFSET OUT5MESS
        JMP     ERRORMESS
```

```
OUT6:       ADD     AL,30H
            MOV     OUT6AL,AL
            MOV     DX,OFFSET OUT6MESS
            JMP     ERRORMESS
LOADPROC:
            MOV     AL,NEWPROC
            CMP     AL,1
            JNZ     LOUT1
            MOV     BP,0
            JMP     RETURNPT
LOUT1:      CMP     AL,2
            JNZ     LOUT2
            MOV     BP,STACKSIZE
            JMP     RETURNPT
LOUT2:      CMP     AL,3
            JNZ     LOUT3
            MOV     BP,2*STACKSIZE
            JMP     RETURNPT
LOUT3:      CMP     AL,4
            JNZ     LOUT4
            MOV     BP,3*STACKSIZE
            JMP     RETURNPT
LOUT4:      ADD     AL,30H
            MOV     LOUT4AL,AL
            MOV     DX,OFFSET LOUT4MESS
            JMP     ERRORMESS
RETURNPT:
            MOV     CURRENTPROC1,AL
            MOV     SP,STACKTBL [BP]
            POP     ES ! POP DI ! POP SI ! POP BP
            POP     DX ! POP CX ! POP BX ! POP AX
            POPF
            STI
            RETF
;
;
;
;THIS  ROUTINE  MAKES  A  SYSYTEM CALL  TO  PUT  OUT  ERROR
;MESSAGES
ERRORMESS:
            MOV     CL,9
            INT     224
            MOV     CL,0
            MOV     DL,1
            INT     224
            RET
;
;
;
```

```
;THE STACK AREAS AND VARIABLE DEFINITIONS

STACKTBL:
          DW      OFFSET STACKTBL+STACKSIZE-22
          RS      STACKSIZE-8
          DW      FALSE
          DW      OFFSET P1
          RS      2
          DW      OFFSET STACKTBL+2*STACKSIZE-22
          RS      STACKSIZE-8
          DW      FALSE
          DW      OFFSET P2
          RS      2
          DW      OFFSET STACKTBL+3*STACKSIZE-22
          RS      STACKSIZE-8
          DW      FALSE
          DW      OFFSET P3
          RS      2
          DW      OFFSET STACKTBL+4*STACKSIZE-22
          RS      STACKSIZE-8
          DW      FALSE
          DW      OFFSET P4
          RS      2
NEWPROC   DB      0
OUT5MESS  DB      FS,ESC,FF,CAN
          DB      'RETURN FROM OUT5. AN UNKNOWN CURRENT PROCEDURE:'
OUT5AL    DB      0,'$',0
OUT6MESS  DB      FS,ESC,FF,CAN
          DB      'RETURN FROM OUT6. STACK OVERFLOW FOR THE PROC.:'
OUT6AL    DB      0,'$',0
LOUT4MESS DB      FS,ESC,FF,CAN
          DB      'RETURN FROM LOUT4. AN UNKNOWN NEW PROCEDURE:'
LOUT4AL   DB      0,'$',0
;
;
;

          END
```

B. AWAIT.PLI

```
/*
Prog Name              : AWAIT.PLI
Date                   : December 83
Written  by            : M. Kadri Ozyurt
For                    : Thesis
Advisor                : Professor Kodres
Purpose                :  This synchronization primitive checks
the  threshold  value for the calling process by  comparing
the  corresponding  threshold  value  with  FOURTHEVC   and
returns  the control either to the calling process,  if its
threshold value is equal to and greater than FOURTHEVC, or
else transfers the control to ARBITER.A86
*/



await:     procedure(i);
  dcl      threshold(0:2) fixed bin(15) external,
           storestatus entry,
           fourthevc fixed bin(15) external ,
           i fixed bin(7);
  if (fourthevc>=threshold(i-1)) then return;
  else call storestatus;
end await;
```

C. SCHEDULE.PLI

```
/*
Prog  Name          : SCHEDULE.PLI
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This   synchronization   primitive
compares   the   threshold  values  corresponding  to   the
processes P1 through P3,  beginning from P1,  to  FOURTHEVC
and  returns the name of the first one which is equal to or
greater than that value.  If non of the processes meet this
conditions then P4 is returned.
*/




schedule: procedure returns (fixed bin(7));
  dcl     threshold(0:2) fixed bin(15) external,
          fourthevc fixed bin(15) external,
          i fixed bin(7);
  do i=0 to 2;
          if (fourthevc>=threshold(i)) then return (i+1);
  end;
  return (4);
end schedule;
```

D. THRESH.PLI

```
/*
Prog  Name          : THRESH.PLI
Date                : December 83
Written by          : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This   synchronization   primitive
receives  a  pointer to the calling process and  increments
the corresponding threshold value by an assigned amount
*/



THRESH: PROC(i);

  DCL THRESHOLD(0:2)FIXED BIN(15) EXTERNAL,
      i FIXED BIN(7);

  IF (i=1) THEN THRESHOLD(0)=THRESHOLD(0) + 1;
  IF (i=2) THEN THRESHOLD(1)=THRESHOLD(1) + 4;
  IF (i=3) THEN THRESHOLD(2)=THRESHOLD(2)+4;
  RETURN;
END THRESH;
```

E. P1.PLI

```
/*
Prog  Name            : P1.PLI
Date                  : December 83
Written  by           : M. Kadri Ozyurt
For                   : Thesis
Advisor               : Professor Kodres
Purpose               : This   process   is   basically   an
infinitive loop.  Once entered, it first call AWAIT.PLI
to  see  if  FOURTHEVC  is equal to  or  greater  than  its
threshold  value.  If it is,  then the control proceeds  to
call  TACTICAL.PLI.  The  last  call  in  the  loop  is  to
THRESH.PLI  to increment its threshold value.  In the  next
iteration,  the  control  will  not  come  back  since  its
threshold value is greater than FORTHEVC.
*/




P1: PROCEDURE;

  DCL AWAIT ENTRY (FIXED BIN(7)),
      THRESH ENTRY (FIXED BIN(7)),
      A FIXED BIN(7),
      TACTICAL ENTRY;

  A=1;
  DO WHILE ('1'B);
          CALL AWAIT (1);
          CALL TACTICAL;
          CALL THRESH (1);
  END /*DO*/;
END P1;
```

F. P2.PLI

```
/*
Prog  Name            : P2.PLI
Date                  : December 83
Written  by           : M. Kadri Ozyurt
For                   : Thesis
Advisor               : Professor Kodres
Purpose               : The purpose of this process is idecti-
cal  to  that of P1.PLI with the exception that the  second
call is to DISPLAY.PLI
*/




P2: PROCEDURE;

  DCL AWAIT ENTRY (FIXED BIN(7)),
      THRESH ENTRY (FIXED BIN(7)),
      A FIXED BIN(7),
      DISPLAY ENTRY;

  A=2;
  DO WHILE ('1'B);
          CALL AWAIT (2);
          CALL DISPLAY;
          CALL THRESH (2);
  END /*DO*/;
END P2;
```

G. P3.PLI

```
/*
Prog  Name           : P3.PLI
Date                 : December 83
Written  by          : M. Kadri Ozyurt
For                  : Thesis
Advisor              : Professor Kodres
Purpose              : The  purpose of this process is  the
same  as P1.PLI with the exception that the second call  is
to STATUS.PLI
*/

P3: PROCEDURE;

  DCL AWAIT ENTRY (FIXED BIN(7)),
      THRESH ENTRY (FIXED BIN(7)),
      A FIXED BIN(7),
      STATUS ENTRY;

  A=3;
  DO WHILE ('1'B);
          CALL AWAIT (3);
          CALL STATUS;
          CALL THRESH (3);
  END /*DO*/;
END P3;
```

H. P4.PLI

```
/*
Prog Name           : P4.PLI
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This process is an infinitive loop
in which there is only one call to IDLE.PLI repeaditively
until an interrupt comes along.
*/




P4: PROCEDURE;

  DCL AWAIT ENTRY (FIXED BIN(7)),
      THRESH ENTRY (FIXED BIN(7)),
      A FIXED BIN(7),
      IDLE ENTRY;

  A=4;
  DO WHILE ('1'B);
          CALL IDLE;
  END /*DO*/;
END P4;
```

APPENDIX E

MISCELLANEOUS ASSEMBLY ROUTINES

A. KEYBOARD.A86

```
;Prog  Name          : KEYBOARD.A86
;Date                : December 83
;Written  by         : M. Kadri Ozyurt
;For                 : Thesis
;Advisor             : Professor Kodres
;Purpose             : This  program  receives  a  formal
;parameter,  KEY,  reads  the  status  of  the  serial  I/O
;interface chip.  If a character has been received from the
;keyboard,  it  reads this character and places it  to  the
;formal parameter.  If there is not a character available
;it  puts  a ascii equivalent of zero into  the  parameter.
;The reason for that is that zero is not used as a keyboard
;command.  The  variables used here are defined in the body
;of  ARBITER.A86



KEYBOARD:
          PUSHF
          CLI
          PUSH    AX
          IN      AL,PORTST
          CMP     AL,RXRDY
          JZ      KEYBOARD1
          IN      AL,PORTIO
          AND     AL,7FH
          JMP     KEYBOARD2
KEYBOARD1:
          MOV     AL,30H
KEYBOARD2:
          MOV     BX,[BX]
          MOV     [BX],AL
          POP     AX
          POPF
          RET
```

B. ATOD.A86

```
;Prog  Name         : ATOD.A86
;Date               : December 83
;Written  by        : M. Kadri Ozyurt
;For                : Thesis
;Advisor            : Professor Kodres
;Purpose            : This program receives two parameters
;It reads the output of the A/D converter specified by the
;second parameter and places it into the first parameter.
;The variables used here are defined in ARBITER.A86



ATOD:
          PUSHF
          CLI
          PUSH      SI
          PUSH      AX
          PUSH      BX
          PUSH      BX
          PUSH      DS
          MOV       BX,[BX]                  ;BX=.ARGUMENT(1)
          MOV       AH,Ø
          MOV       AL,[BX]
          MOV       SI,AX
          MOV       AX,SEGCONV
          MOV       DS,AX
          MOV       BX,OFFCONV
          MOV       AL,[BX+SI]               ;READ A/D PORT
          POP       DS
          POP       BX
          MOV       BX,2[BX]
          MOV       [BX],AL
          POP       BX
          POP       AX
          POP       SI
          POPF
          RET
```

C. RINGBELL.A86

```
;Prog  Name          : RINGBELL.A86
;Date                : December 83
;Written  by         : M. Kadri Ozyurt
;For                 : Thesis
;Advisor             : Professor Kodres
;Purpose             : This program sends a bell character to
;the video terminal. The variables used here are defined in
;the body of ARBITER.A86.



RINGBELL:
           PUSHF
           CLI
           PUSH    AX
           CALL    WAIT
           MOV     AL,BEL
           OUT     PORTIO,AL
           POP     AX
           POPF
           RET
;
;
```

D. WAIT.A86

```
;Prog  Name          : WAIT.A86
;Date                : December 83
;Written  by         : M. Kadri Ozyurt
;For                 : Thesis
;Advisor             : Professor Kodres
;Purpose             : This program program reads the status
;of  the serial I/O chip and waits until the transmitter is
;ready to send characters.



WAIT:
          PUSH    AX
WAIT1:    IN      AL,ØDEH                   ;GET STATUS
          AND     AL,1
          JZ      WAIT1
          POP     AX
          RET
;
;
```

E.SUSPEND.A86

```
;Prog  Name          : SUSPEND.A86
;Date                : December 83
;Written  by         : M. Kadri Ozyurt
;For                 : Thesis
;Advisor             : Professor Kodres
;Purpose             : This program stops the real time clock
;by reseting the interrupt bit of the PSW.




SUSPEND:
          CLI
          RET
```

F. RESUME.A86

```
;Prog  Name          : RESUME.A86
;Date                : December 83
;Written  by         : M. Kadri Ozyurt
;For                 : Thesis
;Advisor             : Professor Kodres
;Purpose             : This program starts the real time clock
;by  reseting the interrupt bit of the PSW.  It then  reset
;the counter to zero.
;



RESUME:
          PUSH    AX
          MOV     AL,CNTR2                    ;RESET COUNTER
          OUT     PORTC,AL
          MOV     AL,CNTRHI
          OUT     COUNT,AL
          MOV     AL,EOI                      ;RESET PIC
          OUT     PIC1,AL
          POP     AX
          STI
          RET
;
;
```

APPENDIX F

DYNAMIC DEBUGGING MODULE LISTINGS

A. LOCALS.AID

```
/*
Prog  Name        : LOCALS.AID
Date              : December 83
Written by        : M. Kadri Ozyurt
For               : Thesis
Advisor           : Professor Kodres
Purpose           : This %include file contains the declara-
tions of the variables used by the dynamic debugging module
*/



  DCL
  BREAKS(0:9) LABEL,
  STOPS(0:9) BIT(1) EXTERNAL,
  (CODE1,CODE2,VALUE,H) FIXED BIN(15),
  BREAKPT FIXED BIN(15) EXTERNAL,
  PUTVARS ENTRY,
  REENTRY ENTRY,
  BREAKPTS ENTRY,
  PROMPTUSER ENTRY (FIXED BIN(7)),
  STORESTATUS ENTRY,
  TACTICAL ENTRY,
  DISPLAY ENTRY,
  IDLE ENTRY,
  STATUS ENTRY,
  CHANGEVA ENTRY (FIXED BIN(15),FIXED BIN(15));
```

B. ERRHAND.AID

```
/*
Prog  Name          : ERRHAND.AID
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This %include file contains six
different types of PL/I ON condtion bodies. Upon
intercepting any raised error condition is displayed
and the control is transfered to PROMPTUSER with a number
that shows which breakpoint has been past. Then the ON
condition body is exited with a non-local goto statement.
At the exit point the control is transfered to REENTRY
which is the dynamic debugging tool. This call to REENTRY
is  protected during the course of normal operation with an
if statement which tests the value of ERRORON.
*/



stops(0)=false;
stops(1)=false;
stops(2)=false;
stops(3)=false;
stops(4)=false;
stops(5)=false;
stops(6)=false;
stops(7)=false;
stops(8)=false;
stops(9)=false;
on error
  begin;

  put list('^Z');                     /*clear screen*/
  put skip list('Error #');

/*this statement gets the code of the error condition*/
  code1=oncode();

/*this call prompts the user with the # of breakpoints past
and asks if the user wants to enter the dynamic debugging
environment*/
  call promptuser(code1);
  if (key='y') ! (key='Y') then goto errorexit;
  else if code1<=127 then do;
          put skip list('The program will be abandoned');
          stop;
  end /*if*/;
```

```
end /*error*/;

on fixedoverflow
  begin;

  put list('^Z');
  put skip list('Fixedoverflow #');
  code1=oncode();
  call promptuser(code1);
  if (key='y') ! (key='Y') then goto errorexit;
  else if code1<=127 then do;
          put skip list('The program will be abandoned');
          stop;
  end /*if*/;

end /*fixedoverflow*/;

on overflow
  begin;

  put list('^Z');
  put skip list('Overflow #');
  code1=oncode();
  call promptuser(code1);
  if (key='y') ! (key='Y') then goto errorexit;
  else if code1<=127 then do;
          put skip list('The program will be abandoned');
          stop;
  end /*if*/;

end /*overflow*/;

on underflow
  begin;

  put list('^Z');
  put skip list('Underflow #');
  code1=oncode();
  call promptuser(code1);
  if (key='y') ! (key='Y') then goto errorexit;
  else if code1<=127 then do;
          put skip list('The program will be abandoned');
          stop;
  end /*if*/;

end /*underflow*/;

on zerodivide
  begin;
```

```
  put list('^Z');
  put skip list('Zerodivide #');
  code1=oncode();
  call promptuser(code1);
  if (key='y') ! (key='Y') then goto errorexit;
  else if code1<=127 then do;
          put skip list('The program will be abandoned');
          stop;
  end /*if*/;

end /*zerodivide*/;

errorexit:
  if erroron then do;
          call reentry();
  end /*if*/;
```

D. REENTRY.PLI

```
/*
Prog  Name          : REENTRY.PLI
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This routine is the "workhorse" of
the dynamic debugging environment. It calls  PUTVARS if
the user wants to see the external variables.Then it
calls CHANGEVA if the user wants to change any variable
with in a loop until no changes are wanted.It then transfers
the control to the breakpoint the user desires.
*/



  reentry:proc external;

/*
  dcl
*/
          %include 'const.inp';
          %include 'globals.inp';
          %include 'locals.aid';

put  skip list('You have entered the interactive  debugging '
                                              ,'environment.');
put  skip  list('You  will  be asked  questions  about  the ',
                                     'control of program flow');
reentry1:
put  skip  list('Do  you want a listing  of  all  variables ',
                                                       '(Y/N)?');
get list(key);
if (key='Y') ! (key='y') then call putvars();
put  skip  list('Do  you want to change the  value  of  any ',
                                              'variable(Y/N)?');
get list (key);
do while  (~((key = 'N') ! (key='n')));
   if (key = 'Y') ! (key='y') then do;
  put  skip  list('Enter the number and the  new  value ',
         ' (-32768<=value<=+32,767) of the veriable you want to');
  put  skip  list('  change  in  integers  seperated  by  a ',
                                                       'comma.');
  on error begin;
     put list ('*** bad entry, try again');
     goto reentry2;
  end /*error*/;
```

```
  on fixedoverflow begin;
          put list('*** too large, try again');
          go to reentry2;
  end /*fixedoverflow*/;
reentry2:
  put skip list('>');
  get list (code1,value);
  revert error;
  revert fixedoverflow;
  if code1>maxvars then
      put list('invalid variable number');
  else
      call changeva(code1,value);
  /*end if*/
  put skip list('Do you want a listing again (Y/N)?');
  get list(key);
  if (key='Y')!(key='y') then call putvars();
  put  skip  list('Do you want to change  another  variable ',
                                                      '(Y/N)?');
      end /*do*/;
    else
  put list('*** bad entry, try again');
    /*end if*/
    get list(key);
end/*do*/;
put skip list('Which breakpoint do you want to transfer the ',
                        'control (0 thru 9, foolowed by return)?');
get list(key);
do while ((rank(key)<48) ! (rank(key)>57));
  put list('*** bad entry, try again');
  put skip list('>');
  get list(key);
end/*do*/;
code1=rank(key)-48;
breakpt=code1;
put  skip  list('Enter the breakpoint you want to  stop  (0 ',
   'thru 9) or any  non-numeral character if you do not  want  to ',
                                                 stop(fol. RET');
get list(key);
if ((rank(key)<48) ! (rank(key)>57)) then do;
  put  skip  list('The program will execute beginning  from ',
                                         'the breakpoint',code1);
  erroron=false;
      end /*do*/;
else
    do;
  code2=rank(key)-48;
  stops(code2)=true;
  put  skip  list('The  program will  execute  between  the ',
                              'breakpts',code1,' and ',code2);
```

```
end /*if*/;

put skip list('Is that what you want(Y/N)?');
get list(key);
if (key='N') ! (key='n') then do;
  put skip list('Do you want another run(Y/N)?');
  get list (key);
  if (key='Y') ! (key='y') then goto reentry1;
end /*if*/;

end reentry;
```

E. PUTVARS.PLI

```
/*
Prog  Name          : PUTVARS.PLI
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This routine puts selected external
variables out with PL/I put edit statement.
*/




putvars:proc external;

  %replace  max_ships by 2,
            true by '1'b,
            false by '0'b;
  /*dcl*/
          %include 'globals.inp';

  i=0;
  j=0;
  put skip list('The listing of all common variables is as ',
                                                   'follows:');
  put skip(2) list('Fixed binary values:');
  put skip edit('(1)seconds=',seconds,'(2)minutes=',minutes,
    '(3)hours=',hours,'(4)wake_ptr=',wake_ptr,'(5)I=',i)(r(format1));
  put skip edit('(6)t_of=',t_of,'(7)target=',target,
    '(8)own=',own,'(9)known=',known,'(10)J=',j)(r(format1));
  put skip edit('(11)t=',t,'(12)t_prime=',t_prime,'(13)currentproc=',
                  currentproc)(r(format1));
  put skip edit('(14)fourthevc=',fourthevc)(a,f(5));
  put skip(2) list('Fixed decimal values:');
  put skip(2) list('Boolean values:');
  put skip edit('(15)engaged=',engaged,'(16)magnified=',magnified,
                '(17)fired=',fired,'(18)erroron=',erroron)
               (a,b(1),col(20),a,b(1),col(41),a,b(1),col(58),a,b(1));
  put skip(2) list('Fixed decimal values:');
  put skip edit('(19)vx_own=',vx_own,'(20)vy_own=',vy_own,
  '(21)vx_target=',vx_target,'(22)vy_target=',vy_target)(r(format2));
  put skip edit('(23)vx_rel=',vx_rel,'(24)vy_rel=',vy_rel,
  '(25)vx_round=',vx_round,'(26)vy_round=',vy_round)(r(format2));
  put skip edit('(27)vr=',vr,'(28)alpha=',alpha,
  '(29)ax=',ax,'(30)bx=',bx)(r(format2));
  put skip edit('(31)cx=',cx,'(32)ay=',ay,'(33)by=',by,'(34)cy=',cy)
                                                      (r(format2));
```

```
        put skip edit('(35)ax_sum=',ax_sum,'(36)bx_sum=',bx_sum,
                      '(37)cx_sum=',cx_sum)(r(format3));
        put skip edit('(38)ay_sum=',ay_sum,'(39)by_sum=',by_sum,
                      '(40)cy_sum=',cx_sum)(r(format3));
        put skip edit('(41)x_at5=',x_at5,'(42)y_at5=',y_at5,
                      '(43)r=',r)(r(format3));
        put skip edit('(44)dx_dt_at5=',dx_dt_at5,'(45)dy_dt_at5=',
                     dy_dt_at5,'(46)dr_dt_at5=',dr_dt_at5)(r(format3));
        put skip(2) list('Character values:');
        put skip edit('(47)key=',key)(a(8),a(1));
        put skip(2)list('Arrays:');
        put skip edit('(48)threshold(0)=',threshold(0),'(49)threshold(1)=',

                    threshold(1),'(50)threshold(2)=',threshold(2))
                    (a,f(5),col(26),a,f(5),col(51),a,f(5));
        put skip(2) list('Data structures:');
        put skip list('ship(1):');
        put skip edit('(51)course=',course(1),'(52)speed=',speed(1),
            '(53)azimuth=',azimuth(1),'(54)range=',range(1))(r(format4));
        put skip edit('(55)x=',x(1),'(56)y=',y(1),'(57)x_aim=',x_aim(1),
                      '(58)y_aim=',y_aim(1))(r(format5));
        put skip edit('(59)count=',count(1),'(60)number=',number(1),
               '(61)ptr=',ptr(1),'(62)link_ship=',link_ship(1))
                                                            (r(format6));
        put skip list('Ship(2):');
        put skip edit('(63)course=',course(2),'(64)speed=',speed(2),
            '(65)azimuth=',azimuth(2),'(66)range=',range(2))(r(format4));
        put skip edit('(67)x=',x(2),'(68)y=',y(2),'(69)x_aim=',x_aim(2),
                      '(70)y_aim=',y_aim(2))(r(format5));
        put skip edit('(71)count=',count(2),'(72)number=',number(2),
               '(73)ptr=',ptr(2),'(74)link_ship=',link_ship(2))
                                                            (r(format6));
        put skip list('Gun:');
        put skip edit('(75)az=',az,'(76)alt=',alt,'(77)x_gun=',x_gun,
                        '(78)y_gun=',y_gun)(r(format5));
        put skip list('Wake(ptr(2)):');
        put skip edit('(79)x_wake=',x_wake(ptr(2)),'(80)y_wake=',
              y_wake(ptr(2)),'(81)link_wake=',link_wake(ptr(2)))
              (a,f(7,1),col(20),a,f(7,1),col(39),a,f(1));
        put skip edit ('(82)dt=',dt)(a,f(4,2));
format1:format(a,f(2),col(16),a,f(2),col(31),a,f(2),col(44),a,f(2),col(59),
                                                       a,f(5));
format2:format(a,f(4,1),col(19),a,f(4,1),col(37),a,f(4,1),col(58),a,
                                                               f(4,1));
format3:format(a,f(4,1),col(24),a,f(4,1),col(47),a,f(4,1));
format4:format(a,f(4,1),col(19),a,f(4,1),col(35),a,f(4,1),col(54),a,
                                                                 f(7,1));
format5:format(a,f(7,1),col(19),a,f(7,1),col(35),a,f(7,1),col(54),a,
                                                                 f(7,1));
format6:format(a,f(1),col(19),a,f(2),col(35),a,f(2),col(54),a,f(2));
end putvars;
```

F. CHANGEVA.PLI

```
/*
Prog Name          : CHANGEVA.PLI
Date               : December 83
Written  by        : M. Kadri Ozyurt
For                : Thesis
Advisor            : Professor Kodres
Purpose            : This routine changes a selected
external variable specified by the parameter passed
*/



changeva:proc(code1,value) external;

  dcl
    (code1,value) fixed bin(15);

    %include 'const.inp';
    %include 'globals.inp';

  if code1=1 then seconds=binary(value,7);
  if code1=2 then minutes=binary(value,7);
  if code1=3 then hours=binary(value,7);
  if code1=4 then wake_ptr=binary(value,7);
  if code1=5 then i=binary(value,15);
  if code1=6 then t_of=binary(value,7);
  if code1=7 then target=binary(value,7);
  if code1=8 then own=binary(value,7);
  if code1=9 then known=binary(value,7);
  if code1=10 then j=binary(value,15);
  if code1=11 then t=binary(value,7);
  if code1=12 then t_prime=binary(value,7);
  if code1=13 then currentproc=binary(value,7);
  if code1=14 then fourthevc=binary(value,15);
  if code1=15 then engaged=bit(value,1);
  if code1=16 then magnified=bit(value,1);
  if code1=17 then fired=bit(value,1);
  if code1=18 then erroron=bit(value,1);
  if code1=19 then vx_own=decimal(value,4,1);
  if code1=20 then vy_own=decimal(value,4,1);
  if code1=21 then vx_target=decimal(value,4,1);
  if code1=22 then vy_target=decimal(value,4,1);
  if code1=23 then vx_rel=decimal(value,4,1);
  if code1=24 then vy_rel=decimal(value,4,1);
  if code1=25 then vx_round=decimal(value,4,1);
  if code1=26 then vy_round=decimal(value,4,1);
  if code1=27 then vr=decimal(value,4,1);
```

```
if code1=28 then alpha=decimal(value,4,1);
if code1=29 then ax=decimal(value,7,2);
if code1=30 then bx=decimal(value,7,2);
if code1=31 then cx=decimal(value,7,2);
if code1=32 then ay=decimal(value,7,2);
if code1=33 then by=decimal(value,7,2);
if code1=34 then cy=decimal(value,7,2);
if code1=35 then ax_sum=decimal(value,7,2);
if code1=36 then bx_sum=decimal(value,7,2);
if code1=37 then cx_sum=decimal(value,7,2);
if code1=38 then ay_sum=decimal(value,7,2);
if code1=39 then by_sum=decimal(value,7,2);
if code1=40 then cy_sum=decimal(value,7,2);
if code1=41 then x_at5=decimal(value,7.2);
if code1=42 then y_at5=decimal(value,7,2);
if code1=43 then r=decimal(value,7,2);
if code1=44 then dx_dt_at5=decimal(value,7,2);
if code1=45 then dy_dt_at5=decimal(value,7,2);
if code1=46 then dr_dt_at5=decimal(value,7,2);
if code1=47 then key=ascii(value);
if code1=48 then threshold(0)=value;
if code1=49 then threshold(1)=value;
if code1=50 then threshold(2)=value;
if code1=51 then course(1)=decimal(value,4,1);
if code1=52 then speed(1)=decimal(value,3,1);
if code1=53 then azimuth(1)=decimal(value,3,0);
if code1=54 then range(1)=decimal(value,5,0);
if code1=55 then x(1)=decimal(value,6,1);
if code1=56 then y(1)=decimal(value,6,1);
if code1=57 then x_aim(1)=decimal(value,6,1);
if code1=58 then y_aim(1)=decimal(value,6,1);
if code1=59 then count(1)=binary(value,7);
if code1=60 then number(1)=binary(value,7);
if code1=61 then ptr(1)=binary(value,7);
if code1=62 then link_ship(1)=binary(value,7);
if code1=63 then course(2)=decimal(value,4,1);
if code1=64 then speed(2)=decimal(value,3,1);
if code1=65 then azimuth(2)=decimal(value,3,0);
if code1=66 then range(2)=decimal(value,5,0);
if code1=67 then x(2)=decimal(value,6,1);
if code1=68 then y(2)=decimal(value,6,1);
if code1=69 then x_aim(2)=decimal(value,6,1);
if code1=70 then y_aim(2)=decimal(value,6,1);
if code1=71 then count(2)=binary(value,7);
if code1=72 then number(2)=binary(value,7);
if code1=73 then ptr(2)=binary(value,7);
if code1=74 then link_ship(2)=binary(value,7);
if code1=75 then az=decimal(value,4,1);
if code1=76 then alt=decimal(value,4,1);
if code1=77 then x_gun=decimal(value,6,1);
```

```
     if code1=78 then y_gun=decimal(value,6,1);
     if code1=79 then x_wake(ptr(2))=decimal(value,6,1);
     if code1=80 then y_wake(ptr(2))=decimal(value,6,1);
     if code1=81 then link_wake(ptr(2))=binary(value,7);
     if code1=82 then dt=float(value,7);

nd changeva;
```

G. BREAKSØ.AID

```
/*
Prog  Name          : BREAKSØ.AID
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This %include file is one of the ten
%include files ,BREAKSØ through BREAKS9, that are used
to insert various parts of the programs to be tested.
They are protected during the normal operation of the
program under test with an if statement. Within the if
statement thereis a call to BREAKPTS .
*/
```

```
REAKS(Ø):
     BREAKPT=Ø;
     IF STOPS(BREAKPT) THEN DO;
             CALL BREAKPTS;
             GOTO BREAKS(BREAKPT);
     END /*IF*/;
```

H. BREAKPTS.PLI

```
/*
Prog  Name        : BREAKPTS.PLI
Date              : December 83
Written  by       : M. Kadri Ozyurt
For               : Thesis
Advisor           : Professor Kodres
Purpose           : This routine prompts the user that the
breakpoint intended to stop has been reached. Then it asks
if the user wants to transfer the control over the dynamic
debuggig  environment.  If  the answer is positive then  it
calls REENTRY where the control stays thereafter.
*/



breakpts:proc external;
  dcl
          stops(Ø:9) bit(1) external,
          breakpt fixed bin(15) external,
          key char(1) external,
          erroron bit(1) external,
          reentry entry;

  stops(breakpt)='Ø'b;
  put skip list('***** breakpoint',breakpt,' *****');
  put skip list('The execution halted and clock stopped.');
  put skip list('Do you want to enter the interactive debugging',
                 'environment(Y/N)?');
  get list(key);
  if (key='Y') ! (key='y') then
          call reentry();
  else
          erroron='Ø'b;
  /*end if*/
end breakpts;
```

I. TIMES.AID

```
/*
Prog  Name           : TIMES.AID
Date                 : December 83
Written  by          : M. Kadri Ozyurt
For                  : Thesis
Advisor              : Professor Kodres
Purpose              : This %include file is inserted to
WAR.PLI to test the execution times of the individual
system routines.
*/




put  skip(2)  list ('Do you want to measure the execution',
,'times of the modules (Y/N)?');
get list (key);
do while ((key='Y') ! (key='y'));
  put skip(2) list('Enter the number of iterations you want ',
                   '(max 32,767).');
  on error begin;
          put list('*** bad entry, try again.');
          goto times1;
  end /*error*/;
  on fixedoverflow begin;
          put list('*** too large, try again.');
          goto times1;
  end /*fixedoverflow*/;
times1:
  put skip(2) list('>');
  get list(h);
  revert error;
  revert fixedoverflow;
  put skip(2) list('Get ready for time check. The modules',
                               ' will execute ',h,' times.');
  do i=1 to 4;
          put skip(2) list('Ready!! Press any key to start'
                          ,'the  time check of the module');
          if i=1 then put list (' IDLE.');
          else if i=2 then put list(' STATUS.');
          else if i=3 then put list(' TACTICAL.');
          else put list(' DISPLAY.');
          get list(key);
          do j=1 to h while (i=1); call idle; end;
          do j=1 to h while (i=2); call status; end;
          do j=1 to h while (i=3); call tactical; end;
          do j=1 to h while (i=4); call display; end;
```

```
          put skip(2)list('The end of the execution .');
          put skip(2)list('Enter the time  measured in ',
                                                   'seconds.');
          on error begin;
                  put list('*** bad entry, try again');
                  go to times2;
          end /*error*/;
          on fixedoverflow begin;
                  put list('*** bad entry,try again.');
                  goto times2;
          end /*fixedoverflow*/;
times2:
          put skip(2) list('>');
          get list(j);
          revert error;
          revert fixedoverflow;
          begin;
            dcl duration float;
            duration=float(j)/float(h);
            put skip(2) list('The execution time of the ',
                    'module is',duration,' iterations/sec');
          end;
  end /*do*/;
  put skip(2) list('Do you want another run (Y/N)?');
  get list(key);
end /*do*/;
```

APPENDIX G

A SAMPLE SUBROUTINE TESTING

```
/*
Prog  Name          : P.PLI

Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This program is written to test
individual procedures in an interactive manner. At each
iteration  new  values  are asked.  The PL/I  ON  condition
bodies  are  used  to  intercept  any  inadvertantly  wrong
entries. The endless loop can be terminated either ^C or
^Z  from  the  terminal.  In this  particular  example  the
procedure DRAW inside the body of DISPLAY.PLI is tested by
making it external for the test purposes.
*/


p:proc options(main);

dcl
  (u,v)(0:10) fixed bin(15),
  (i,x,y)fixed bin(15),
  rub char(1) external,
  d fixed bin(7),
  draw entry ((0:10)fixed bin(15),(0:10)fixed bin(15)
                  ,fixed bin(7));


on error begin;
    put skip list('*** bad value, try again');
    goto reentry;
end;

on fixedoverflow begin;
    put skip list('*** too large, try again');
    goto reentry;
end;
```

```
reentry:
rub=ascii(127);
do while('1'b);
    put list('^/^_^X');
    put skip list('enter x and y');
    put skip list('>');
    get list(x,y);
    put list('^Z');
    call gen(x,y,u,v);
    d=1;
    call draw(u,v,d);
    call delay;
    d=0;
    call draw(u,v,d);
end;

delay:proc;
  dcl (i,j) fixed bin(15);

  do i=1 to 30000;
          do j=1 to 2;
          end;
  end;
end;

gen:proc(x,y,u,v);

  dcl
          (u,v)(0:10) fixed bin(15),
          (x,y) fixed bin(15);

u(0)=x+8;           v(0)=y;
u(1)=x;             v(1)=y+8;
u(2)=x-8;           v(2)=y;
u(3)=x;             v(3)=y-8;
u(4)=x+8;           v(4)=y;
u(5)=-1;            v(5)=-1;

end;

end p;
```

APPENDIX H

A SAMPLE PROGRAM TESTING

```
/*
Prog  Name          : IDLE.PLI
Date                : December 83
Written  by         : M. Kadri Ozyurt
For                 : Thesis
Advisor             : Professor Kodres
Purpose             : This is the testing version of the
procedure  IDLE.PLI under the dynamic debugging module.
After the correct result from the test had been taken,
the final version of the procedure was made simply removing
the segment of code in between the comment lines. In order
to test the program, an interactive main procedure as in
Appendix G was written.
*/




IDLE:PROCEDURE EXTERNAL;



/*
  DCL
*/
      %INCLUDE 'CONST.INP';
      %INCLUDE 'GLOBALS.INP';


/******* DEBUG AID *******/
      %INCLUDE 'LOCALS.AID';
      %INCLUDE 'ERRHAND.AID';
      %INCLUDE 'BREAKS0.AID';
/******** END AID ********/

      DO D=0 TO 5;
          CALL ATOD (D,ARG(D));
      END /*DO*/;
```

```
/*at  this  point  the A/D output values are  fixed  bin(7)
values.  The  following  sequence converts those  to  fixed
decimal values*/

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS1.AID';
/******* END AID ********/

      COURSE(OWN)=ARG(0);
      SPEED(OWN)=ARG(4);
      COURSE(KNOWN)=ARG(2);
      SPEED(KNOWN)=ARG(3);
      AZ=ARG(1);
      ALT=ARG(5);

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS2.AID';
/******* END AID ********/

/*the  following sequence converts A/D values to real  time
values by using appropriate proportionality constants*/
      COURSE(OWN) = COURSE(OWN) * K;
      COURSE(KNOWN) = COURSE(KNOWN) * K;
      AZ = AZ * K;
      IF COURSE(OWN)<0.0 THEN
          COURSE(OWN) = COURSE(OWN) + TWO_PI ;
      IF COURSE(KNOWN)<0.0 THEN
          COURSE(KNOWN) = COURSE(KNOWN) + TWO_PI ;
      IF AZ<0.0 THEN
          AZ = AZ + TWO_PI;
       IF ALT>90.0 THEN
          ALT = 90.0;

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS3.AID';
/******* END AID ********/

          SPEED(OWN) = SPEED(OWN)/L;
          SPEED(KNOWN) = SPEED(KNOWN) / L;

/*ownship speed computations*/
      VX_OWN = SPEED(OWN) * SIND(COURSE(OWN));
      VY_OWN = SPEED(OWN) * COSD(COURSE(OWN));

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS4.AID';
/******* END AID ********/
```

```
/*when not have fired, the following makes the ballistic
computations*/
      IF ~ FIRED THEN
         BEGIN;
          T_OF = 2.0 * VM * SIND(ALT) / G;

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS5.AID';
/******* END AID ********/

          VR = VM * COSD(ALT);
          R = VR * T_OF;

/****** DEBUG AID *******/
     %INCLUDE 'BREAKS6.AID';
/******* END AID ********/

          X_AIM(OWN) = R * SIND(AZ);
          Y_AIM(OWN) = R * COSD(AZ);
          X_GUN = 0.0;
          Y_GUN = 0.0;

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS7.AID';
/******* END AID ********/

          VX_ROUND = VR * SIND(AZ);
          VY_ROUND = VR * COSD(AZ);

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS8.AID';
/******* END AID ********/

          T = T_OF;

      END /*IF*/;

/****** DEBUG AID ******/
     %INCLUDE 'BREAKS9.AID';
/******* END AID ********/

END IDLE;
```

LIST OF REFERENCES

1. Digital Engineering, Inc, User's Manual, RG-512 Retro-Graphics Card for the ADM-3A Computer Terminal, 1981

2. Rector, R. and Alexy, G., The 8086 Book, Osborne, McGraw-Hill, 1980

3. Digital Research Corporation, CP/M-86 Operating Systems Guide, 1981

4. Digital Research Corporation, PL/I-86 Manual, 1983

5. Lamie, E.L., PL/I Programming, Wadsworth Publishing Co., 1982

6. Digital Research Corporation, PL/I-80 Applications Guide, 1980

7. Intel Corporation, iSBC 86/12A Single Board Computer Hardware Reference Manual, 1979

8. Lear Siegler, Inc., ADM 3A Dumb Terminal Video Display Unit User Reference Manual, Anaheim, 1981

9. Intel Corporation, MCS-86 Assembly Language Reference Manual, Santa Clara, 1978

10. Kodres, U., Class Notes - CS.3550, Naval Postgraduate School, 1983

BIBLIOGRAPHY

Deitel, H.M., An Introduction to Operating Systems, Addison-Wesley Publishing Company, 1982

Kersh, T. B., Signal Processor Interface Simulation of the AN/SPY-1A Radar Controller, Master's Thesis, Naval Postgraduate School, 1983

Maclennan, B., Programming Language Design Principles, Naval Postgraduate School, 1982

Selcuk, Z., Kim, K.C., and Bozkurt, D., Air Surveillance System Simulation; CS-3550 Class Project, Naval Postgraduate School, 1983

Tenenbaum, A.M. and Augenstein, M. J., Data Structures Using Pascal, Prentice-Hall, Inc., 1981

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defence Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Computer Science, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
4.	Professor U. Kodres, Code 52KR Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
5.	Hakan Ozyurt 145 Sok. 6/6 B. Blok Kopru - Izmir TURKEY	1
6.	Mike Williams, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
7.	LCDR R.B. Kurth, Code 52K Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
8.	O.R. Chambers Systems analyst Corrections Division 2575 Center St. N.E. Salem, Oregon 97310	1
9.	Dz. K. Komutanligi Okullar ve Kurslar Dairesi Bakanliklar - Ankara TURKEY	5

10. Deniz Harb Okulu 1
Kutuphanesi
Heybeliada - Istanbul TURKEY

11. Istanbul Teknik Universitesi 1
Kutuphanesi
Istanbul TURKEY

12. Bogazici Universitesi 1
Kutuphanesi
Istanbul TURKEY
TURKEY

13. Orta Dogu Teknik Universitesi 1
Kutuphanesi
Ankara TURKEY

14. M. Kadri Ozyurt 1
145 Sok 6/6 Kopru
Izmir TURKEY

207336

Thesis
0999 Ozyurt
c.1
Real-time applications in multiprocessor systems.